uc3m | Universidad Carlos III de Madrid

CC BY NC SA

OpenCourseWare

Database

2.2. Query language. SQL language

Lourdes Moreno López
Paloma Martínez Fernández
José Luis Martínez Fernández
Rodrigo Alarcón García

HULAT
Human Language & Accessibility Technologies

Content

- Introduction
- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Introduction to trigger

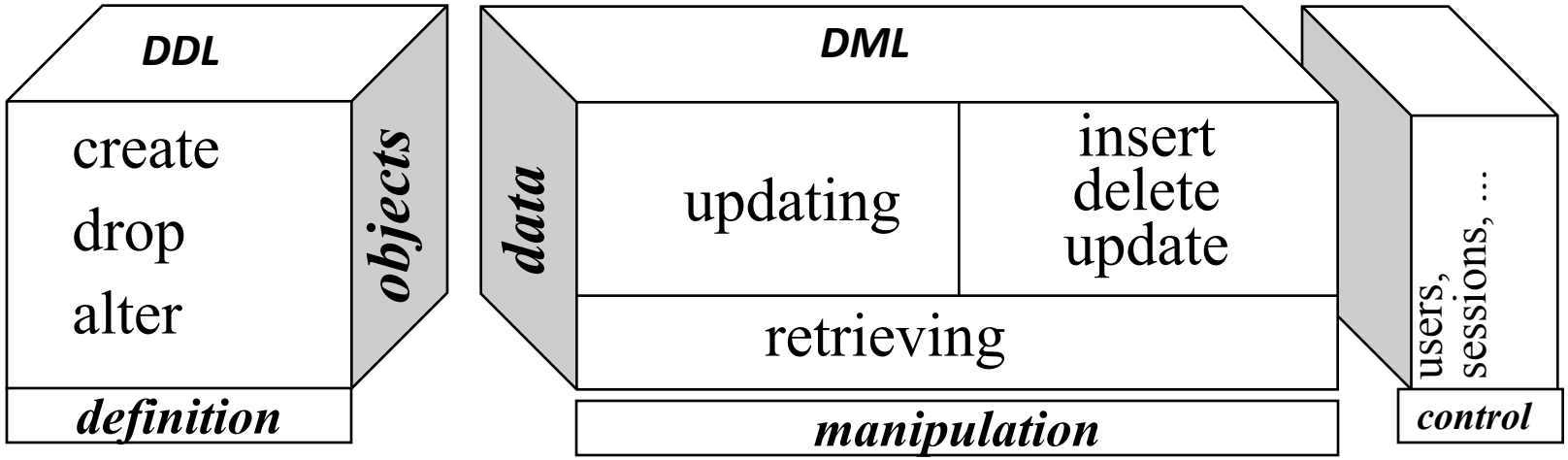
SQL

Introduction

- Structured Query Language (SQL)
- It is a language for the development of the relational model
- In 1986, a standard for SQL was defined by the American National Standards Institute (ANSI) and was subsequently adopted in 1987 as an international standard by the International Organization for Standardization (ISO, 1987).
  - Releases: SQL/89, SQL/92, SQL/1999, …
- More than one hundred DBMSs now support SQL

Introduction

- a Data Definition Language (DDL) for defining the database structure and controlling access to the data
- a Data Manipulation Language (DML) for retrieving and updating data



Data Definition Language (DDL)

Data Definition Language (DDL)

- CREATE: To create a table
- ALTER: To delete, add or even modify the columns of an existing table, to add constrains, ..
- DROP: To drop a table

CREATE table

- The CREATE TABLE command is used to specify a new relation by giving it a name and specifying its attributes and initial constraints.
- The attributes are specified first, and each attribute is given a name, a data type to specify its domain of values, and possibly attribute constraints, such as NOT NULL.

```
CREATE TABLE table_name(
  column1 datatype,
  column2 datatype,
  column3 datatype,
  .....
  columnN datatype,
  PRIMARY KEY( one or more columns )
);
```

CREATE table

- Example: PROJECT (Pname,Pnumber,Plocation, Dnum)

```
CREATE TABLE PROJECT
(PNAME VARCHAR(15) NOT NULL,
PNUMBER NUMBER ,
PLOCATION VARCHAR(15),
CONSTRAINT PK_PROJECT PRIMARY KEY (PNUMBER));
```

CREATE table

- The key, entity integrity, and referential integrity constraints can be specified within the CREATE TABLE statement after the attributes are declared, or they can be added later using the ALTER TABLE command.

```
CREATE TABLE PROJECT
(PNAME CHAR(15) NOT NULL,
PNUMBER NUMBER ,
PLOCATION CHAR(15),
DNUM NUMBER NUMBER NULL ,
CONSTRAINT PK_PROJECT PRIMARY KEY
(PNUMBER),
CONSTRAINT FK_PROJECT_DNUM FOREIGN
KEY (DNUM) REFERENCES
DEPARTMENT(DNUMBER)
);
```

```
CREATE TABLE PROJECT
(PNAME CHAR(15) NOT NULL,
PNUMBER NUMBER ,
PLOCATION CHAR(15),
DNUM NUMBER NOT NULL ,
CONSTRAINT PK_PROJECT PRIMARY KEY
(PNUMBER));


ALTER TABLE PROJECT ADD CONSTRAINT
FK_PROJECT_DNUM FOREIGN KEY (DNUM)
REFERENCES DEPARTMENT(DNUMBER);
```

Attribute Data Types

- The basic data types available for attributes include:
  - character, string
  - numeric
  - date
  - boolean
  - time

https://docs.oracle.com/database/121/SQLRF/sql_elements001.htm#SQLRF0021

Data Types character string

	SQL standard	Oracle
Fixed -length character string	CHARACTER(n) CHAR(n)	CHAR(n)
Variable -length character string	CHARACTER VARYING(n) VARCHAR (n)	VARCHAR2(n)
Fixed -length character string in the national character set.	NATIONAL CHARACTER(n) NATIONAL CHAR(n) NCHAR(n)	NCHAR(n)
Variable -length character string in the national character set.	NATIONAL CHARACTER VARYING(n) NATIONAL CHAR VARYING(n) NCHAR VARYING(n)	NVARCHAR2(n)

Data Types numeric

	SQL standard	Oracle
Small integers (2 bytes)	SMALLINT	
Integers (2 bytes)	INTEGER INT	
Decimal precision using integer		NUMBER(n)
Floating point number	FLOAT DOUBLE DOUBLE PRECISSION REAL	NUMBER
Fixed point number	NUMERIC(m,d) DECIMAL(m,d)	NUMBER(m,d)

Data Types
date

	SQL standard	Oracle
Date	DATE	DATE
Date and time	TIMESTAMP	TIMESTAMP
Intervals	INTERVAL YEAR TO MONTH INTERVALE DAY TO SECOND	INTERVAL YEAR TO MONTH INTERVALE DAY TO SECOND

Data Types
boolean and binaries

	SQL standard	Oracle
Boolean (logical)	BOOLEAN BOOL	
Binaries	BIT BIT VARYING VARBIT(n)	

Data Types large object

	SQL standard	Oracle
character large object	CHARACTER LARGE OBJECT CLOB	LONG (deprecated) CLOB
Binary large object	BINARY LARGE OBJECT BLOB	RAW (deprecated) LONG RAW (deprecated) BLOB BFILE

Domains

- It is possible to specify the data type of each attribute directly, or a domain can be declared
- The domain name can be used with the attribute specification
- This makes it easier to change the data type for a domain that is used by numerous attributes

```
CREATE DOMAIN SSN_TYPE AS CHAR(9);
```

Constraints

- Basic constraints that can be specified in SQL as part of table creation
  - Attribute Constraints and Attribute Defaults
  - Key and Referential Integrity Constraints
  - Constraints on Tuples Using CHECK

Attribute Constraints and Attribute Defaults

- SQL allows NULLs as attribute values
- A constraint **NOT NULL** may be specified if **NULL** is not permitted for a particular attribute.
  - This is always implicitly specified for the attributes that are part of the primary key of each relation
  - but it can be specified for any other attributes whose values are required not to be NULL

```
CREATE TABLE DEPARTMENT
(DNAME VARCHAR(15) NOT NULL ,
DNUMBER NUMBER
MGRSSN CHAR(9) NOT NULL);
```

Attribute Constraints and Attribute Defaults

- It is also possible to define a default value for an attribute by appending the clause **DEFAULT** <value> to an attribute definition
- The default value is included in any new tuple if an explicit value is not provided for that attribute

```
Dno NUMBER NOT NULL DEFAULT 1,
```

Attribute Constraints and Attribute Defaults

- Another type of constraint can restrict attribute or domain values using the **CHECK** clause following an attribute or domain definition

```
Dnumber NUMBER NOT NULL CHECK (Dnumber >
0 AND Dnumber < 21);
```

- The CHECK clause can also be used in conjunction with the CREATE DOMAIN statement

```
CREATE DOMAIN D_NUM AS INTEGER CHECK
(D_NUM > 0 AND D_NUM < 21);
```

Key and Referential Integrity Constraints

- Keys and referential integrity constraints are special clauses within the CREATE TABLE.
- The **PRIMARY KEY** clause specifies one or more attributes that make up the primary key of a relation.
- If a primary key has a single attribute, the clause can follow the attribute directly.

```
CREATE TABLE PROJECT
(PNAME CHAR(15) NOT NULL,
PNUMBER NUMBER ,
PLOCATION CHAR(15),
DNUM NUMBER NOT NULL ,
CONSTRAINT PK_PROJECT PRIMARY KEY (PNUMBER),
CONSTRAINT FK_PROJECT_DNUM FOREIGN KEY (DNUM)
REFERENCES DEPARTMENT(DNUMBER)
);
```

Key and Referential Integrity Constraints

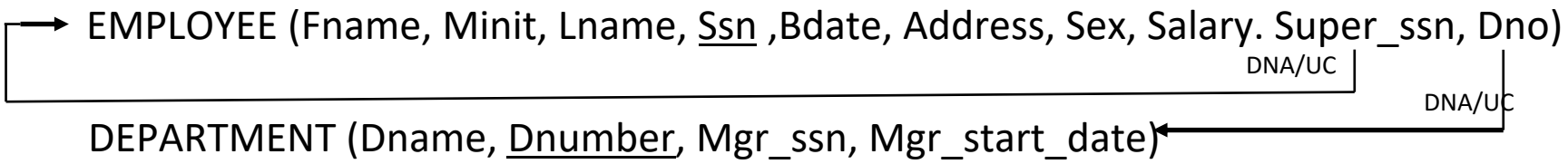
- The **UNIQUE** clause specifies alternate (unique) keys or candidate keys.
- The UNIQUE clause can also be specified directly for a unique key if it is a single attribute.

```
Dname VARCHAR(15) UNIQUE,
```

Key and Referential Integrity Constraints

- Referential integrity is specified via the **FOREIGN KEY** clause
- As we discussed, a referential integrity constraint can be violated when tuples are inserted or deleted, or when a foreign key or primary key attribute value is updated
- The default action that SQL takes for an integrity violation is to reject with RESTRICT option, or SET NULL, CASCADE, and SET DEFAULT
- An option must be qualified with either ON DELETE or ON UPDATE

Key and Referential Integrity Constraints




```
CREATE TABLE EMPLOYEE
(FNAME CHAR(15) NOT NULL ,
MINIT CHAR,
LNAME VARCHAR(15) NOT NULL ,
SSN CHAR(9) NOT NULL ,
BDATE DATE,
ADDRESS VARCHAR(30) ,
SEX CHAR,
SALARY DECIMAL(10,2) ,
SUPERSSN CHAR(9) NOT NULL ,
DNO NUMBER NOT NULL ,
CONSTRAINT PK_EMPLOYEE  PRIMARY KEY (SSN)
CONSTRAINT FK_EMPLOYEE_SUPERSSN FOREIGN KEY (SUPERSSN) REFERENCES
EMPLOYEE(SSN),
CONSTRAINT FK_EMPLOYEE_DNO FOREIGN KEY (DNO) REFERENCES DEPARTMENT(DNUMBER)
);
```

Constraints on Tuples Using CHECK

- Other table constraints can be specified through additional **CHECK** clauses at the end of a CREATE TABLE statement
- These can be called row-based constraints because they apply to each row individually and are checked whenever a row is inserted or modified

```
CREATE TABLE EMPLOYEE
(FNAME CHAR(15) NOT NULL ,
MINIT CHAR,
LNAME VARCHAR(15) NOT NULL ,
SSN CHAR(9) NOT NULL ,
BDATE DATE,
ADDRESS VARCHAR(30) ,
SEX CHAR,
SALARY DECIMAL(10,2) ,
SUPERSSN CHAR(9) NOT NULL ,
DNO NUMBER NOT NULL ,
CONSTRAINT PK_EMPLOYEE  PRIMARY KEY (SSN)
CONSTRAINT FK_EMPLOYEE_SUPERSSN FOREIGN KEY
(SUPERSSN) REFERENCES EMPLOYEE(SSN),
CONSTRAINT FK_EMPLOYEE_DNO FOREIGN KEY
(DNO) REFERENCES DEPARTMENT(DNUMBER)
CONSTRAINT CH_SEX CHECK (SEX IN ('F', 'M')
);
```

Schema

- SQL schema groups together tables, types, constraints, views, domains, and other constructs that belong to the same database application
- An SQL schema is identified by a schema name and includes an authorization identifier to indicate the user or account who owns the schema.

```
CREATE SCHEMA COMPANY AUTHORIZATION 'Jsmith';
```

- Database installations have a default schema, so when a user connects and logs in to that database installation

Catalog

- A catalog is a named collection of schemas
- A catalog always contains a special schema called INFORMATION_SCHEMA, which provides information on all the schemas in the catalog and all the element descriptors in these schemas.

DROP table

- The **DROP** command can be used to drop named schema elements, such as SCHEMA , tables, domains, types, or constraints
- There are two drop behavior options: CASCADE and RESTRICT

```
DROP TABLE EMPLOYEE CASCADE CONSTRAINTS;

DROP TABLE DEPARTMENT CASCADE CONSTRAINTS;
```

ALTER table

- The definition of a base table or of other named schema elements can be changed by using the **ALTER** command
- The possible alter table actions include:
  - adding or dropping a column (attribute)
  - changing a column definition
  - adding or dropping table constraints.

ALTER table

```
ALTER TABLE COMPANY.EMPLOYEE ADD COLUMN Job
VARCHAR(12);

ALTER TABLE COMPANY.EMPLOYEE DROP COLUMN Address
CASCADE;

ALTER TABLE COMPANY.DEPARTMENT ALTER COLUMN Mgr_ssn
SET DEFAULT '333445555';

ALTER TABLE COMPANY.EMPLOYEE DROP CONSTRAINT
EMPSUPERFK CASCADE;

ALTER TABLE WORKS_ON ADD CONSTRAINT FK_WORKS_ON_PNO
FOREIGN KEY (PNO) REFERENCES PROJECT(PNUMBER) ON
DELETE CASCADE;

ALTER TABLE WORKS_ON ADD CONSTRAINT FK_WORKS_ON_ESSN
FOREIGN KEY (ESSN) REFERENCES EMPLOYEE(SSN) ON
DELETE CASCADE;
```

Data Manipulation Language (DML)

- **Insert, delete, update**
- Retrieving (queries)

The INSERT, DELETE, UPDATE Commands

- **INSERT** is used to add a single tuple (row) to a relation (table)
- We must specify the relation name and a list of values for the tuple. The values should be listed in the same order in which the corresponding attributes were specified in the CREATE TABLE command

```
INSERT INTO EMPLOYEE VALUES ( 'Richard',
'K', 'Marini', '653298653', '1962-12-30',
'98 Oak Forest, Katy, TX', 'M', 37000,
'653298653', 4 );
```

The INSERT, DELETE, UPDATE Commands

- A second form of the INSERT statement allows the user to specify explicit attribute names that correspond to the values provided in the INSERT command

```
INSERT INTO EMPLOYEE (Fname, Lname, Dno,
Ssn) VALUES ('Richard', 'Marini', 4,
'653298653');
```

- Attributes with NULL allowed or DEFAULT values are the ones that can be left out.

The INSERT, DELETE, UPDATE Commands

- A DBMS that fully implements SQL should support and enforce all the integrity constraints that can be specified in the DDL

```
INSERT INTO EMPLOYEE (Fname, Lname, Ssn, Dno)
VALUES ('Robert', 'Hatcher', '980760540', 2);
```

(it is rejected if referential integrity checking is provided by DBMS.)

```
U2A: INSERT INTO EMPLOYEE (Fname, Lname, Dno)
VALUES ('Robert', 'Hatcher', 5);
```

(it is rejected if NOT NULL checking is provided by DBMS.)

The INSERT, DELETE, UPDATE Commands

- The **DELETE** command removes tuples from a relation.
- It includes a WHERE clause, to select the tuples to be deleted.
- Tuples are explicitly deleted from only one table at a time. However, the deletion may propagate to tuples in other relations if referential triggered actions are specified in the referential integrity constraints of the DDL

```
DELETE FROM EMPLOYEE
WHERE Lname = 'Brown';
```

The INSERT, DELETE, UPDATE Commands

- The **UPDATE** command is used to modify attribute values of one or more selected tuples
- However, updating a primary key value may propagate to the foreign key values of tuples in other relations if such a referential triggered action is specified in the referential integrity constraints of the DDL
- An additional SET clause specifies the attributes to be modified and their new values

The INSERT, DELETE, UPDATE Commands

```
UPDATE PROJECT
SET Plocation = 'Bellaire', Dnum = 5
WHERE Pnumber = 10;
```

Data Manipulation Language (DML)

- Insert, delete, update
- **Retrieving (queries)**

Retrieving (queries)

- SQL has one basic statement for retrieving information from a database: SELECT statement

```
SELECT <attribute list>
FROM <table list>
WHERE <condition>;
```

- where
  - FROM clause gives the relation(s) to which the query refers
  - WHERE clause is a <condition> which tuples must satisfy in order to match the query
  - SELECT clause tells which attributes of the tuples matching the condition are produced as part of the answer (* means that the entire tuple is produced)

Retrieving (queries)

This topic is organized as:

1. Simple Queries: Retrieving data from a single relation
2. Complex Queries: Retrieving data from several tables.

SELECT- SIMPLE QUERIES

Example: Find the information about movies from Disney in 1990

MOVIE (title, year, length, inColor, studioName)

```
SELECT *
FROM MOVIE
WHERE studioName= `Disney´ AND
year=1990;
```

SELECT- SIMPLE QUERIES

1. For each tuple
2. Appy WHERE clause to the tuple (replace the attributes in WHERE by the value in the tuple´s component for that attribute)
3. Evaluate condition: if true, the components appearing in SELECT clause are produced as one tuple of the answer

Title	Year	Length	inColor	studioName
Star Wars	1977	124	true	Fox
Mighty Ducks	1991	104	true	Disney
Wayne´s World	1992	95	true	Paramount
Pretty Woman	1990	1990	true	Disney

SELECT- SIMPLE QUERIES

- Output

WHERE `Disney´ = `Disney´ AND 1990 = 1990

Title	Year	Length	inColor	studioName
Pretty Woman	1990	1990	true	Disney

SELECT- SIMPLE QUERIES

- PROJETION (COLUMNS)
- SELECTION (ROWS)
- AGGREGATION (GROUPS)

SELECT- SIMPLE QUERIES-PROJECTION

PROJECTION (to select some columns)

1. Find the birth date and address of the employee(s)

```
SELECT Ssn, Bdate, Address
FROM EMPLOYEE
```

2. Find all data from employees

```
SELECT *
FROM EMPLOYEE
```

SELECT- SIMPLE QUERIES-PROJECTION

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

Fuente: Fundamentals of Database Systems, 7 ed., Ramez Elmasri Y Shamkant B. Navathe

SELECT- SIMPLE QUERIES-PROJECTION

- Alias: attributes can be renamed as output

```
SELECT Bdate AS Date_of_birth,  Address
FROM EMPLOYEE
```

Date_of_Birth	Address
1965-01-09	731 Fondren, Houston, TX
……	…….
……	…….

SELECT- SIMPLE QUERIES-PROJECTION

- Some arithmetic operators can be used.

The standard arithmetic operators for addition (+), subtraction (–), multiplication (*), and division (/) can be applied to numeric values or attributes with numeric datatypes

Employees with salary increased by 10%

```
SELECT Fname, Lname, 1.1 * E.Salary AS Increased_sal
FROM EMPLOYEE
```

SELECT- SIMPLE QUERIES-PROJECTION

- Duplicate values
  - A query with SELECT DISTINCT eliminates duplicates, whereas a query with SELECT ALL does not.
  - Specifying SELECT with neither ALL nor DISTINCT equivalent to SELECT ALL.

```
SELECT ALL Salary
FROM EMPLOYEE;

SELECT DISTINCT Salary
FROM EMPLOYEE;
```

SELECT- SIMPLE QUERIES- PROJECTION

```
SELECT Title as Name, length*0.016667 AS length, `hrs.
´AS inHours FROM MOVIE
```

Output:

Title	Year	Length	inColor	studioName
Star Wars	1977	124	true	Fox
Mighty Ducks	1991	104	true	Disney
Wayne´s World	1992	95	true	Paramount
Pretty Woman	1990	110	true	Disney

Name	Length	inHours
Star Wars	2.06	hrs.
Mighty Ducks	1.73	hrs.
Wayne´s World	1.58	hrs.
Pretty Woman	1.83	hrs.

SELECT- SIMPLE QUERIES-PROJECTION

- Ordering the output
  - If we want the tuples produced by a query be presented in sorted order

```
ORDER BY <list_of_attributes>
```

  - The order is by default ascending but we can specified ASC/DESC

```
SELECT Dname, Mgr_ssn, Mgr_start_date
FROM DEPARTMENT
ORDER BY Dname
```

SELECT- SIMPLE QUERIES-SELECTION

- SELECTION (which tuples we want as output)
  - A missing WHERE clause indicates no condition on tuple selection; hence, all tuples of the relation specified in the FROM clause qualify and are selected for the query result

```
SELECT Ssn, Dname
FROM EMPLOYEE, DEPARTMENT;
```

  - The expressions that may follow WHERE include conditional expressions like used in any programming language:

  - Six common operators: =, <>, <, >, <=, >= to compare constants and attributes

SELECT- SIMPLE QUERIES-SELECTION

- String-valued constants are denoted by surrounding them with single quoted
- Numeric constants, integers and reals are also allowed
- The result of a comparation is a Boolean value (TRUE or FALSE)
- Boolean values may be combined by the LOGICAL OPERATORS:
  - AND
  - OR
  - NOT
- Remember precedence; NOT->AND OR and the use of parenthesis.

SELECT- SIMPLE QUERIES-SELECTION

1. Find the birth date and address of the employee(s) whose name is 'John B. Smith'

```
SELECT Bdate, Address
FROM EMPLOYEE
WHERE Fname = 'John' AND Minit = 'B' AND Lname =
'Smith';
```

1. Find the name and address of all employees who work for the department 5

```
SELECT Fname, Lname, Address
FROM EMPLOYEE,
WHERE Dno = 5;
```

(see EMPLOYEE table at Company DB)

SELECT- SIMPLE QUERIES-SELECTION

1.

Bdate	Address
1965-01-09	731 Fondren, Houston, TX

2.

Fname	Lname	Address
John	Smith	731 Fondren, Houston, TX
Franklin	Wong	638 Voss, Houston, TX
Ramesh	Narayan	975 Fire Oak, Humble, TX
Joyce	English	5631 Rice, Houston, TX

SELECT- SIMPLE QUERIES-SELECTION

Find the date of birth and address of John B. Smith

```
SELECT Bdate AS Date_of_birth,  Address
FROM EMPLOYEE
WHERE Fname = 'John' AND Minit = 'B' AND
Lname = 'Smith';
```

Date_of_Birth	Address
1965-01-09	731 Fondren, Houston, TX

SELECT- SIMPLE QUERIES-SELECTION

```
SELECT Title as Name, length*0.016667 AS length, `hrs.
´AS inHours
FROM MOVIE
WHERE studioName= `Disney´ AND year=1990;
```

Title	Year	Length	inColor	studioName
Star Wars	1977	124	true	Fox
Mighty Ducks	1991	104	true	Disney
Wayne´s World	1992	95	true	Paramount
Pretty Woman	1990	110	true	Disney

Output:

Name	Length	inHours
Pretty Woman	1.983334	hrs.

SELECT- SIMPLE QUERIES-SELECTION

- Comparison of strings
  - Two strings are equal if they are the same sequence of characters
  - If we compare with "lesser than"/greater than" operators alphabetically order (for instance, 'Martin'< 'Martínez')
  - SQL allows to compare strings on the basis of a simple pattern match:
    - s LIKE p
  - Partial strings are specified using two reserved characters:
    - % replaces an arbitrary number of zero or more characters (wildcard)
    - underscore (_) replaces a single character

SELECT- SIMPLE QUERIES-SELECTION

Find the employees with an address containing `Houston, TX´

```
SELECT Fname, Lname
FROM EMPLOYEE
WHERE Address LIKE '%Houston, TX%';
```

SELECT- SIMPLE QUERIES-SELECTION

Comparison of strings (special case!)

Find the movies with a possessive (´s) in their titles

```
SELECT title
FROM MOVIE
WHERE title LIKE `%´´s%´
```

- The apostrophe can not represent itself (SQL convention is ´´ to represent it)

SELECT- SIMPLE QUERIES-SELECTION

- Comparing Dates and Times
  - Variety of formats (5/14/1948 or 14 may 1948)
  - A date is represented by the keyword DATE followed by a quoted string

    DATE '1948-05-14'
  - Check the datatype format in DBMS

    Oracle: to_date('05/22/1988', 'mm/dd/yyyy')

SELECT- SIMPLE QUERIES-SELECTION

- Others
  - Expression **IN (….)**: Check if an expression is in a list of values
  - Find the names of Employees living in Houston or Stafford

```
SELECT Fname, Lname
FROM EMPLOYEE
WHERE Dlocation IN (`Houston´, `Stafford´)
```

SELECT- SIMPLE QUERIES-SELECTION

- Others
  - Expression **IS NULL/ IS NOT NULL**

    =NULL doesn´t work, you need to use IS NULL

  - Find the names of employees without supervisor

```
SELECT Fname, Lname
FROM EMPLOYEE
WHERE Super_ssn IS NULL
```

SELECT- SIMPLE QUERIES-SELECTION

- Others
  - Expression **BETWEEN min AND max**

Find the names of employees with a 30.000 € >=salary <= 40.000€

```
SELECT Fname, Lname
FROM EMPLOYEE
WHERE (Salary BETWEEN 30000 AND 40000)
```

SELECT- SIMPLE QUERIES-AGGREGATION

AGGREGATION

- Class of operations that aggregate values in a column.
  1. Forming a single value from the list of values in the column (ex, “sum of salaries of employees”, “average of salaries of employees”)
  2. Also grouping the tuples of a relation according to some criterion (such as the value in some other column) and then aggregate within each group.

SELECT- SIMPLE QUERIES-AGGREGATION

- **Aggregation operators**: SQL provides 5 operator that apply to a column and produce some summary or aggregation of that column.
  1. SUM, the sum of the values in this column
  2. AVG, the average of values in this column
  3. MIN, the least value in the column
  4. MAX, the greatest value in the column
  5. COUNT (column), the number of values (including duplicates unless if they are eliminated with DISTINCT)
  6. COUNT (*), the number of tuples

SELECT- SIMPLE QUERIES-AGGREGATION

- Find the average salary of all employees in the Company DB

```
SELECT AVG(Salary) "Average salary of all employees"
FROM EMPLOYEE
```

Ecode	Ename	Edept	Salary	Date_of_Birth	Phone
A1	Pablo Montero	14	30.000	10-11-67	6543
A2	Beatriz Cristobal	13	45.000	20-9-68	6577
A3	J.Luis Martín	11	22.000	25-6-77	6433
A4	Almudena López	13	32.000	4-5-60	6422
A5	Angel Vallejo	14	40.000	15-4-72	6321
A6	Pedro García	11	28.000	12-3-70	6323

- Outcome: "Average salary of all employees" 31953,66971

SELECT- SIMPLE QUERIES-AGGREGATION

- Find the number of employees of the company

```
SELECT COUNT(*) "Number of employees"
FROM EMPLOYEE
```

Ecode	Ename	Edept	Salary	Date_of_Birth	Phone
A1	Pablo Montero	14	30.000	10-11-67	6543
A2	Beatriz Cristobal	13	45.000	20-9-68	6577
A3	J.Luis Martín	11	22.000	25-6-77	6433
A4	Almudena López	13	32.000	4-5-60	6422
A5	Angel Vallejo	14	40.000	15-4-72	6321
A6	Pedro García	11	28.000	12-3-70	6323

  - Outcome: "Number of employees" 6

SELECT- SIMPLE QUERIES-AGGREGATION

- **Grouping**
  - When aggregation operators are used, the default group is the overall relation.
  - If you want to divide the table into groups, **GROUP BY** is used to specify the column (s) by which you want to group following the **WHERE** clause.
  - The grouping attributes should also appear in the SELECT clause.

Find the number of employees and average salary per department

```
SELECT Dno, COUNT (*) AS Employees, AVG
(Salary)
FROM EMPLOYEE
GROUP BY Dno;
```

SELECT- SIMPLE QUERIES-AGGREGATION

```
SELECT Dno, COUNT (*) AS Employees, AVG (Salary)
FROM EMPLOYEE
GROUP BY Dno;
```

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

Outcome:

Dno	Employees	AVG(Salary)
1	1	55000
4	3	31000
5	4	33250

3 groups: Departments 1, 4 and 5

Fuente: Fundamentals of Database Systems, 7 ed., Ramez Elmasri Y Shamkant B. Navathe

SELECT- SIMPLE QUERIES-AGGREGATION

```
SELECT studioName, COUNT(*)
FROM MOVIE
GROUP BY StudioName;
```

Title	Year	Length	inColor	studioName
Star Wars	1977	124	true	Fox
Mighty Ducks	1991	104	true	Disney
Wayne´s World	1992	95	true	Paramount
Pretty Woman	1990	110	true	Disney

Fox, Disney, Paramount

Outcome:

studioName	COUNT(*)
Fox	1
Disney	2
Paramount	1

SELECT- SIMPLE QUERIES-AGGREGATION

- **HAVING**
  - Suppose that we did not wish to include all the groups as results
  - If we want to choose groups based on some aggregated property of the group itself then GROUP BY should be followed by HAVING clause with the condition about the group.

Find the number of employees and average salary per department for those departments with more than 33000 $ of average salary

```
SELECT Dno, COUNT (*) AS Employees, AVG(Salary)
FROM EMPLOYEE
GROUP BY Dno
HAVING AVG(Salary) >= 33000
```

SELECT- SIMPLE QUERIES

SUMMARY SIMPLE QUERIES

```
SELECT <attribute and function list>
FROM <table list>
[ WHERE <condition> ]
[ GROUP BY <grouping attribute(s)> ]
[ HAVING <group condition> ]
[ ORDER BY <attribute list> ];
```

SELECT- COMPLEX QUERIES-NESTED QUERIES

Nested queries

A SELECT (subquery) inside a SELECT (query)

The result of the subquery is used in a condition in WHERE clause

Two possibilities:

1. When the subquery returns a single value then WHERE clause of query uses '=' operator and the subquery only retrieves a single row.
2. When the subquery returns several values then WHERE clause uses 'IN' operator that allows in the condition to compare to a list of values.

SELECT- COMPLEX QUERIES-NESTED QUERIES

Project numbers of projects of the department where Smith employees are assigned as managers

```
SELECT DISTINCT Pnumber
FROM PROJECT
WHERE Pnumber IN
  ( SELECT Pnumber
  FROM PROJECT, DEPARTMENT, EMPLOYEE
  WHERE Dnum = Dnumber AND
  Mgrssn = Ssn AND Lname = 'Smith' )
```

SELECT- COMPLEX QUERIES-NESTED QUERIES

Last and first names of employees that work in the same department that employee with SSN 123456789

```
SELECT Fnane, Lname
FROM EMPLOYEE
WHERE Dno =
  (SELECT Dno
  FROM EMPLOYEE
  WHERE Ssn='123456789')
```

SELECT- COMPLEX QUERIES-NESTED QUERIES

Last and first names of employees that work in the same department that employee with SSN 123456789

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

Fuente: Database System Concepts, 5th Ed. Silberschatz, Korth and Sudarshan. 2006

SELECT- COMPLEX QUERIES-NESTED QUERIES

Last and first names of employees that work in the same department that employee with SSN 123456789

```
SELECT DISTINCT Essn
FROM WORKS_ON
WHERE (Pno, Hours) IN
          (SELECT Pno, Hours
          FROM WORKS_ON
          WHERE Essn = '123456789' );
```

SELECT- COMPLEX QUERIES

- Using simple queries we can only select data from ONE table
- If we need data from SEVERAL tables we need to combine then.
- Two ways:

1. JOIN operation that combines rows from two or more tables
2. Set-theoretic operations: UNION, INTERSECTION, DIFFERENCE

SELECT- COMPLEX QUERIES-JOIN

SQL has a way to couple relations in one query:

(a) list each table in the FROM clause

(b) Then SELECT and WHERE clauses can refer to the attributes of any of the relations in the FROM clause

JOIN of two tables: The join condition compares two columns, each from a different table. To execute a join, the DBMS combines pairs of rows, each containing one row from each table, for which the join condition evaluates to TRUE. The columns in the join conditions need not also appear in the select list.

- Use the FK in the join combination
- Attributes can be disambiguated using Table_name.Attribute

SELECT- COMPLEX QUERIES-JOIN

- **Join operations** take two relations and return as a result another relation.
- These additional operations are typically used as subquery expressions in the **from** clause
- **Join condition** – defines which tuples in the two relations match, and what attributes are present in the result of the join.
- **Join type** – defines how tuples in each relation that do not match any tuple in the other relation (based on the join condition) are treated.

SELECT- COMPLEX QUERIES-JOIN

Join types
inner join
left outer join
right outer join
full outer join

Join Conditions
natural
on <predicate>
using $(A_1, A_1, \ldots, A_n)$

SELECT- COMPLEX QUERIES-JOIN

- **INNER JOIN:** INNER JOIN returns all rows that match the ON condition. (INNER JOIN is also called JOIN).
  - **NATURAL JOIN:** If the tables have columns with the same name, you can use NATURAL JOIN instead of JOIN.
- **LEFT OUTER JOIN:** LEFT OUTER JOIN returns all rows from the left table with matching rows from the right table. Rows without a match are filled with NULLs. (LEFT OUTER JOIN is also called LEFT JOIN)
- **RIGHT OUTER JOIN:** RIGHT OUTER JOIN returns all rows from the right table with matching rows from the left table. Rows without a match are filled with NULLs. (RIGHT OUTER JOIN is also called RIGHT JOIN).
- **FULL OUTER JOIN:** FULL OUTER JOIN returns all rows from the left table and all rows from the right table. It fills the non-matching rows with NULLs. (FULL OUTER JOIN is also called FULL JOIN)

SELECT- COMPLEX QUERIES-Join Examples

STUDENT (<u>student_id</u>, student_name, syllabus_id)

SYLLABUS (<u>syllabus_id</u>, syllabus_name)

STUDENT

student_id	student_name	syllabus_id
1	John	3
2	Mery	NULL
3	Chris	1
4	James	4
5	Laura	1

SYLLABUS

syllabus_id	syllabus_name
1	database
2	calculus
3	statistics
4	programming

SELECT- COMPLEX QUERIES-Join Examples

INNER JOIN is also called JOIN

```
SELECT *
FROM student INNER JOIN syllabus
ON student. syllabus_id = syllabus.syllabus_id
```

RESULT

student_id	student_name	syllabus_id	syllabus_id	syllabus_name
5	Laura	1	1	database
3	Chris	1	1	database
1	John	3	3	statistics
4	James	4	4	programming

SELECT- COMPLEX QUERIES-Join Examples

```
SELECT *
FROM student NATURAL JOIN syllabus
```

RESULT

syllabus_id	student_id	student_name	syllabus_name
1	5	Laura	database
1	3	Chris	database
3	1	John	statistics
4	4	James	programming

SELECT- COMPLEX QUERIES-Join Examples

```
SELECT *
FROM student LEFT OUTER JOIN syllabus
ON student. syllabus_id = syllabus.syllabus_id
```

LEFT OUTER JOIN is also called LEFT JOIN

RESULT

student_id	student_name	syllabus_id	syllabus_id	syllabus_name
5	Laura	1	1	database
3	Chris	1	1	database
1	John	3	3	statistics
4	James	4	4	Programming
1	Mery	NULL	NULL	NULL

SELECT- COMPLEX QUERIES-Join Examples

```
SELECT *
FROM student RIGHT OUTER JOIN syllabus
ON student. syllabus_id = syllabus.syllabus_id
```

RIGHT OUTER JOIN is also called RIGHT JOIN

RESULT

student_id	student_name	syllabus_id	syllabus_id	syllabus_name
5	Laura	1	1	database
3	Chris	1	1	database
1	John	3	3	statistics
4	James	4	4	Programming
NULL	NULL	NULL	2	calculus

SELECT- COMPLEX QUERIES-Join Examples

```
SELECT *
FROM student FULL OUTER JOIN syllabus
ON student. syllabus_id = syllabus.syllabus_id
```

FULL OUTER JOIN is also called FULL JOIN

RESULT

student_id	student_name	syllabus_id	syllabus_id	syllabus_name
5	Laura	1	1	database
3	Chris	1	1	database
1	John	3	3	statistics
4	James	4	4	Programming
NULL	NULL	NULL	2	calculus
1	Mery	NULL	NULL	NULL

SELECT- COMPLEX QUERIES-JOIN EXAMPLES

- EXAMPLE: Relation *loan*

loan_number	*branch_name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

loan

customer_name	*loan_number*
Jones	L-170
Smith	L-230
Hayes	L-155

borrower

Fuente: Database System Concepts, 5th Ed. Silberschatz, Korth and Sudarshan. 2006

SELECT- COMPLEX QUERIES-Join EXAMPLES

- *loan* **inner join** *borrower* **on** *loan.loan_number = borrower.loan_number*

loan_number	*branch_name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

loan

customer_name	*loan_number*
Jones	L-170
Smith	L-230
Hayes	L-155

borrower

Fuente: Database System Concepts, 5th Ed. Silberschatz, Korth and Sudarshan. 2006

SELECT- COMPLEX QUERIES-Join EXAMPLES

- *loan* **inner join** *borrower* **on** *loan.loan_number = borrower.loan_number*

loan_number	*branch_name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

loan

customer_name	*loan_number*
Jones	L-170
Smith	L-230
Hayes	L-155

borrower

loan_number	*branch_name*	*amount*	*customer_name*	*loan_number*
L-170	Downtown	3000	Jones	L-170
L-230	Redwood	4000	Smith	L-230

Fuente: Database System Concepts, 5th Ed. Silberschatz, Korth and Sudarshan. 2006

SELECT- COMPLEX QUERIES-Join EXAMPLES

- *loan* **natural inner join** *borrower*

loan_number	*branch_name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

loan

customer_name	*loan_number*
Jones	L-170
Smith	L-230
Hayes	L-155

borrower

Fuente: Database System Concepts, 5th Ed. Silberschatz, Korth and Sudarshan. 2006

SELECT- COMPLEX QUERIES-Join EXAMPLES

- *loan* **natural inner join** *borrower*

loan_number	*branch_name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

loan

customer_name	*loan_number*
Jones	L-170
Smith	L-230
Hayes	L-155

borrower

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith

Fuente: Database System Concepts, 5th Ed. Silberschatz, Korth and Sudarshan. 2006

SELECT- COMPLEX QUERIES-Join EXAMPLES

- *loan* **left outer join** *borrower* **on** *loan.loan_number = borrower.loan_number*

loan_number	*branch_name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

loan

customer_name	*loan_number*
Jones	L-170
Smith	L-230
Hayes	L-155

borrower

Fuente: Database System Concepts, 5th Ed. Silberschatz, Korth and Sudarshan. 2006

SELECT- COMPLEX QUERIES-Join EXAMPLES

- *loan* **left outer join** *borrower* **on** *loan.loan_number = borrower.loan_number*

loan_number	*branch_name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

loan

customer_name	*loan_number*
Jones	L-170
Smith	L-230
Hayes	L-155

borrower

loan_number	*branch_name*	*amount*	*customer_name*	*loan_number*
L-170	Downtown	3000	Jones	L-170
L-230	Redwood	4000	Smith	L-230
L-260	Perryridge	1700	*null*	*null*

Fuente: Database System Concepts, 5th Ed. Silberschatz, Korth and Sudarshan. 2006

SELECT- COMPLEX QUERIES-Join EXAMPLES

- *loan* **natural right outer join** *borrower*

loan_number	*branch_name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

loan

customer_name	*loan_number*
Jones	L-170
Smith	L-230
Hayes	L-155

borrower

Fuente: Database System Concepts, 5th Ed. Silberschatz, Korth and Sudarshan. 2006

SELECT- COMPLEX QUERIES-Join EXAMPLES

- *loan* **natural right outer join** *borrower*

loan_number	*branch_name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

loan

customer_name	*loan_number*
Jones	L-170
Smith	L-230
Hayes	L-155

borrower

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-155	*null*	*null*	Hayes

Fuente: Database System Concepts, 5th Ed. Silberschatz, Korth and Sudarshan. 2006

SELECT- COMPLEX QUERIES-Join EXAMPLES

- *loan* **full outer join** *borrower* **using** (*loan_number*)

loan_number	*branch_name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

loan

customer_name	*loan_number*
Jones	L-170
Smith	L-230
Hayes	L-155

borrower

Fuente: Database System Concepts, 5th Ed. Silberschatz, Korth and Sudarshan. 2006

SELECT- COMPLEX QUERIES-Join EXAMPLES

- *loan* **full outer join** *borrower* **using** (*loan_number*)

loan_number	*branch_name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

loan

customer_name	*loan_number*
Jones	L-170
Smith	L-230
Hayes	L-155

borrower

loan_number	*branch_name*	*amount*	*customer_name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*
L-155	*null*	*null*	Hayes

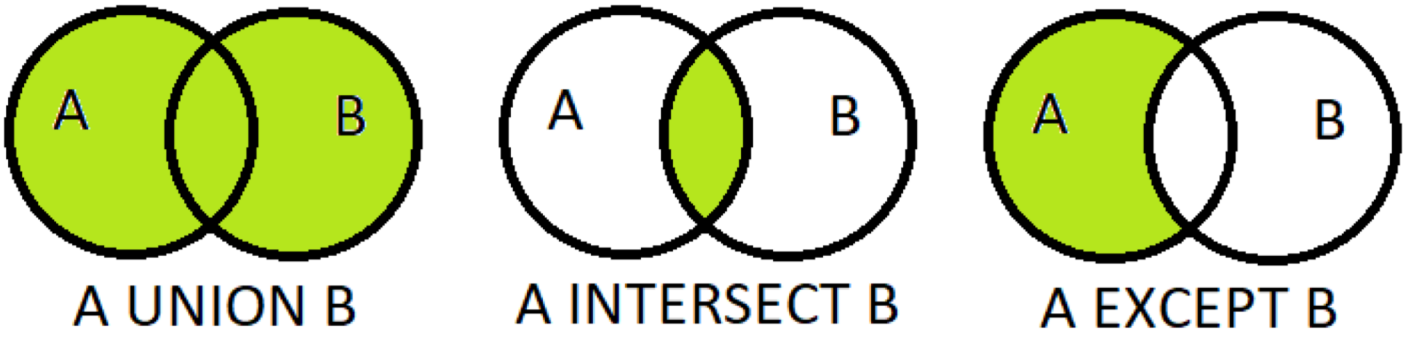
Fuente: Database System Concepts, 5th Ed. Silberschatz, Korth and Sudarshan. 2006

SELECT- COMPLEX QUERIES-Set-theoretic operations

- Set operations are used to combine the results of two or more queries into a single result.
- The combined queries must return the same number of columns and compatible data types. The names of the corresponding columns can be different.
  - **UNION**
  - **INTERSECT**
  - **EXCEPT**

SELECT- COMPLEX QUERIES-Set-theoretic operations

- **UNION** combines the results of two result sets and removes duplicates.
- **INTERSECT** returns only rows that appear in both result sets.
- **EXCEPT** returns only the rows that appear in the first result set but do not appear in the second result set.



SELECT- COMPLEX QUERIES-Set-theoretic operations-Examples

STUDENT

id	name	campus
1	John	Leganés
2	Mery	Getafe
3	Chris	Leganés
4	James	Getafe
5	Laura	Leganés

PROFESSOR

id	name	campus
1	Lourdes	Leganés
2	Pedro	Getafe
4	Mathias	Getafe
5	Laura	Leganés

Fuente: Database System Concepts, 5th Ed. Silberschatz, Korth and Sudarshan. 2006

SELECT- COMPLEX QUERIES-Set-theoretic operations-Examples

SELECT name
FROM student WHERE campus='Leganes'
UNION
SELECT name
FROM professor WHERE campus = 'Leganes';

name
John
Chris
Laura
Lourdes

SELECT- COMPLEX QUERIES-Set-theoretic operations-Examples

SELECT name
FROM student WHERE campus='Leganes'
INTERSECT
SELECT name
FROM professor WHERE campus = 'Leganes';

name
Laura

SELECT- COMPLEX QUERIES-Set-theoretic operations-Examples

SELECT name
FROM student WHERE campus='Leganes'
EXCEPT
SELECT name
FROM professor WHERE campus = 'Leganes';

name
John
Chris
Lourdes

INTRODUCTION TO TRIGGER

Introduction

- **Triggers** are blocks of Procedural Language SQL code associated with a table and that are executed automatically in reaction to a specific DML operation (INSERT, UPDATE or DELETE) on that table.
- For example (project 1.2) : Implementation of the semantic constraint `SERIES.Num_seasons`
  - when a row is inserted into the `SEASON` table, then `SERIES.Num_seasons` is incremented by one.

Syntax for TRIGGER SQL Statement

- **Creation: (activated when created)**

```
CREATE [OR REPLACE] TRIGGER <name_trigger>
{BEFORE | AFTER} event ON referenced table
[FOR EACH ROW [WHEN event_condition]]
body_trigger;
```

- **Elimination:** `DROP TRIGGER name_trigger;`
- **Activation / Deactivation:**

```
ALTER TRIGGER name_trigger {DISABLE | ENABLE};
ALTER TABLE name_trigger
{ENABLE | DISABLE} ALL TRIGGERS;
```

Syntax for TRIGGER SQL Statement Components

```
CREATE [OR REPLACE] TRIGGER <name_trigger>
{BEFORE | AFTER} event ON referenced table
[FOR EACH ROW [WHEN event_condition]]
body_trigger;
```

- **Trigger Name:** They follow the same nomenclature standards as other identifiers in the BD
- **Replace:** Used to overwrite an existing trigger
- **Before / After:** Instant execution of the trigger with respect to the event
- **Event:** DML order type on a table that triggers trigger activation {INSERT | DELETE | UPDATE}

Syntax for TRIGGER SQL Statement Components

```
CREATE [OR REPLACE] TRIGGER <name_trigger>
{BEFORE | AFTER} event ON referenced table
[FOR EACH ROW [WHEN event_condition]]
body_trigger;
```

- **Level**:
  - FOR EACH ROW: row-level triggers. They are activated once for each row affected by the event
  - FOR EACH STATEMENT: triggers with order level. They are activated only once (before or after the statement).
- **When**: It only makes sense at the row level. The condition is evaluated (true or false).

Records :old y :new

- A trigger with row-level is executed on each row in which the event occurs, it uses the records :old and :new
- :old and :new are records that allow us to access the data in the current row

event	:old	:new
INSERT	NULL	new values
UPDATE	stored values	new values
DELETE	stored values	NULL

- Example: Compare if the salary has increased by more than 25% of the old salary:

```
:NEW.salary > (:OLD.salary*1.25)
```

Records :old y :new UPDATE

- When we are making a modification (UPDATE) of a row we can refer to the value before being modified (: OLD) and the value after the modification (: NEW).
- Example:

```
UPDATE PROJECT SET Plocation = 'Bellaire', Dnum= 5
WHERE Pnumber = 10;
```

:NEW

Pname	Pnumber	Plocation	Dnum
ProductX	10	Houston	5

:OLD

Records :old y :new INSERT

- When entering new values (INSERT) we can reference only the new value (: NEW).
- Example:

```
INSERT INTO EMPLOYEE (Fname, Lname, Dno, Ssn)
VALUES ('Richard', 'Marini', 4, '653298653');
```

:NEW

Records :old y :new DELETE

- When we delete (DELETE) we can reference only the old value (:OLD).
- Example:

```
DELETE FROM EMPLOYEE WHERE Lname = 'Smith';
```

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX

:OLD

Conditional predicates

- When a trigger is created for more than one DML operation, a conditional predicate can be used in the statements that make up the trigger that indicates what type of operation or statement the trigger has triggered.
- Predicate triggers (boolean), used to determine what operation is being performed on a trigger.
- These conditional predicates are as follows:
  - **Inserting**: Returns true when the trigger has been triggered by an INSERT
  - **Deleting**: Returns true when the trigger has been triggered by a DELETE
  - **Updating**: Returns true when the trigger has been triggered by an UPDATE

Examples COMPANY

- Example (1): Create a trigger on the EMPLOYEE table that does not allow an employee to be a supervisor of more than five employees.

```
CREATE OR REPLACE TRIGGER control_superssn
BEFORE INSERT ON employee
FOR EACH ROW
DECLARE num NUMBER;
BEGIN
  SELECT count(*) INTO num FROM employee WHERE
  superssn = :new.superssn;
  IF (num > 4)
   THEN raise_application_error (-
   10001,:new.superssn || 'cannot supervise more
   than 5'');
  END IF;
END;
```

The trigger generates a error, the INSERT operation is aborted

Examples COMPANY

- Example (2): Create a trigger to prevent an employee's salary from increasing by more than 20%

```
CREATE OR REPLACE TRIGGER increaseSalary
BEFORE UPDATE OF salary ON employee
FOR EACH ROW
 BEGIN
 IF :NEW.salary > :OLD.salary*1.20
  THEN raise_application_error (-
  10002,:new.Salary||'the salary cannot be
  increased more than 20');
END IF;
END;
```

Examples COMPANY

- Example (3): Create a trigger to avoid that, when inserting an employee, the employee and his supervisor can belong to different departments.

```
CREATE OR REPLACE TRIGGER same_dno
BEFORE INSERT ON employee
FOR EACH ROW
DECLARE dno_superssn NUMBER;
BEGIN
   IF (:NEW.superssn IS NOT NULL) THEN SELECT dno INTO
   dno_superssn FROM employee WHERE ssn=:NEW.superssn;
   IF (dno_superssn <> :NEW.dno)
   THEN raise_application_error (-10003,:NEW.dno ||' An
   employee and his supervisor cannot belong to different
   departments');
   END IF;
   END IF;
END;
```

Examples COMPANY

- Example (4): Create a trigger to prevent that, when inserting an employee, the sum of the salaries of the employees belonging to the department of the inserted employee exceeds 10,000 euros.

Examples COMPANY

- Example (4)

```
CREATE OR REPLACE TRIGGER sumDno
BEFORE INSERT ON employee
FOR EACH ROW
DECLARE sum NUMBER;
BEGIN
   SELECT SUM(salary) INTO sum FROM employee WHERE dno
   =:NEW.dno;
   sum := sum + :NEW.salary;
   IF (sum > 10000) THEN raise_application_error (-
   10004,:NEW.dno||' The sum of salaries cannot exceed
   10000');
END IF;
END;
```

References

- ORACLE Doc: Using Triggers
https://docs.oracle.com/cd/E17781_01/appdev.112/e18147/tdddg_triggers.htm#TDDDG51000

References, bibliography

- Fundamentals of Database Systems by Elmasri, Navathe 7th ed. 2017
- Connolly, Thomas M, Begg, Carolyn E. Database systems: a practical approach to design, implementation, and management. Addison Wesley. 2015
- Database System Concepts, 5th Ed. Silberschatz, Korth and Sudarshan. 2006
- Oracle Database Online Documentation 12c Release 1 (12.1), https://docs.oracle.com/database/121/index.htm