LLENGUATGE SQL - LLENGUATGE DE MANIPULACIÓ DE DADES (DML - DATA MANIPULATION LANGUAGE)

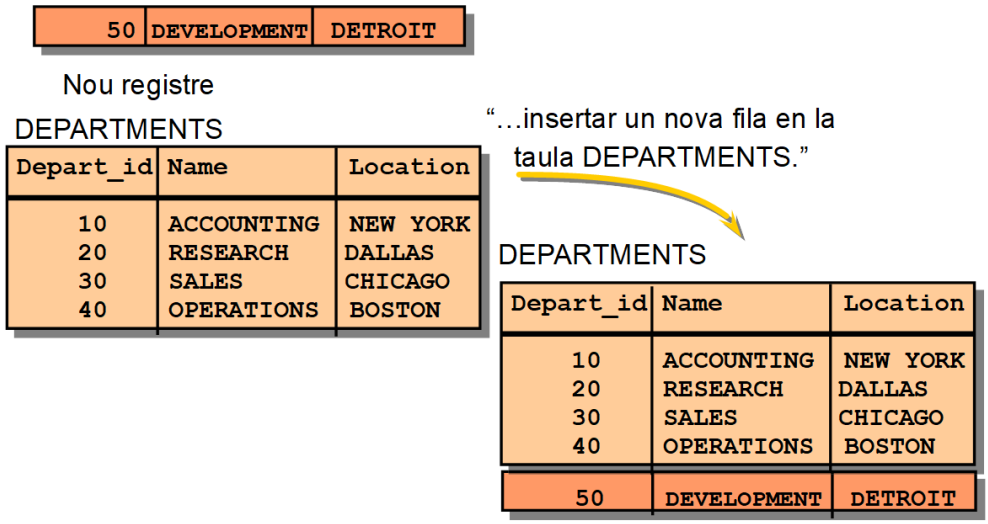
DML (Data Manipulation Language) és el llenguatge SQL que conté totes les sentències per a la manipulació de les dades d'una taula.

Una sentència DML s'executa quan:

- Afegeix nous registres a una taula.
- Modifica registres existents.
- Elimina registres existents.

Una **transacció** consisteix en una col·lecció de sentències DML que formen una unitat lògica de treball.

AFEGIR NOVES FILES EN UNA TAULA



- Afegim noves files a una taula, mitjançant la sentència **INSERT**

```
INSERT INTO   table [(column [, column...])]
VALUES        (value [, value...])
[RETURNING *|column, column…];
```

Aspectes a tenir en compte en la inserció d'una nova fila:

- Inserir una nova fila contenint els valors per a cada columna. Opcionalment, es poden llistar les columnes a la clàusula INSERT
- Col·locar els valors en l'ordre que les columnes tenen a la taula per defecte (segons com es va crear amb CREATE TABLE).
- Escriure els valors de dades de tipus caràcter i data entre cometes simples.

> Exemple: Inserir un nou departament.

```
INSERT INTO departments (department_id, department_name, location_id)
VALUES (50, 'Development', 8);
```

INSERCIÓ DE VALORS NULL

Tenim dues formes d'inserir files amb columnes amb valor Null:

- Mètode implícit: Ometre la columna a la llista.

Exemple: Inserim un nou departament deixant amb valor Null la columna location_id

```
INSERT INTO departments (department_id, department_name)
VALUES (60, 'Operations');
```

- Mètode explícit: Especifiquem el valor clau NULL per a la columna.

Exemple: Inserim un nou departament deixant amb valor Null la columna location_id

```
INSERT INTO departments (department_id, department_name, location_id)
VALUES (60, 'Operations', NULL);
```

Podem utilitzar qualsevol funció de fila única en la sentència INSERT.

Exemple: Inserim un nou empleat amb la data del sistema com a data de contractació.

```
INSERT INTO employees (employee_id, first_name, last_name, hire_date,
department_id, job_id)
VALUES (150, 'James', 'Green', CURRENT_DATE, 1, 8);
```

INSERCIÓ DE FILES AMB COLUMNES SERIAL (AUTONUMÈRIQUES)

En funció del SGBD tindrem diverses maneres d'inserir files amb columnes autoincrementals, habitualment ometrem la columna i el SGBD s'encarregarà d'assignar-li el valor numèric corresponent. En el cas de Postgresql, però, les columnes tipus SERIAL admeten valors introduïts per l'usuari, indicant el valor de forma explícita (si són claus primàries haurem de vigilar amb no violar cap regla d'integritat). Si volem que sigui el SGBD Postgresql qui assigni de forma automàtica el valor de la columna SERIAL haurem d'utilitzar la clàusula **DEFAULT**.

Exemple: Donem d'alta un nou departament, deixant que sigui el SGBD qui assigni el valor del camp department_id

```
INSERT INTO departments (department_id, department_name, location_id)
VALUES (DEFAULT, 'IT', 10);
```

INSERCIÓ DE FILES MÚLTIPLES

Postgresql permet inserir diverses files en una mateixa sentència INSERT.

```
INSERT INTO   table [(column [, column...])]
VALUES        (value [, value...]),
              (value [, value...]),
…
[RETURNING *|column, column…];
```

```
INSERT INTO
    links (url, name)
VALUES
    ('https://www.google.com','Google'),
    ('https://www.yahoo.com','Yahoo'),
    ('https://www.bing.com','Bing');
```

CREACIÓ DE TAULES AMB DADES A PARTIR D'UNA CONSULTA

En el Tema 3 vàrem veure com crear noves taules amb la sentència CREATE TABLE. Aquestes taules es creen des de 'zero' i sense dades, és a dir, només creem l'estructura de la taula, a partir d'aquí inserirem dades amb la sentència INSERT.

Tenim l'opció però, de crear taules a partir d'una consulta, i per tant amb les dades resultants d'aquesta.

CREACIÓ DE TAULES AMB SELECT .. INTO

Crea una nova taula i insereix les files retornades per la sentència SELECT.

```
SELECT select_list
INTO [ TABLE ] new_table_name
FROM
     table_name
WHERE
     search_condition;
```

La taula resultant tindrà el nombre de columnes, tipus de dada i nom que les columnes especificades a la consulta.

> Exemple: Creació de la taula emp_40 amb les dades dels empleats del departament 40. L'ordenació afectarà a la l'ordre en que s'inseriran les files.
>
> ```
> SELECT employee_id, first_name, last_name, hire_date, salary, department_id,
> job_id, manager_id
> INTO emp_40
> FROM employees
> ```

```
WHERE department_id = 40
ORDER BY departemnt_id;
```

CREACIÓ DE TAULES A PARTIR D'UNA SUBCONSULTA

Consisteix a crear una taula i inserir files combinant CREATE TABLE amb la clàusula **AS** subquery.

```
CREATE TABLE table
        [column(, column...)]
AS subquery;
```

Han de coincidir el nombre de columnes especificades amb les de la subconsulta. Haurem de definir columnes amb noms de columna i valors per defecte.

Exemple1: Creació de la taula emp_40 amb les dades dels empleats del departament 40. L'ordenació afectarà a la l'ordre en que s'inseriran les files.

```
CREATE TABLE emp_40 AS
SELECT employee_id, first_name, last_name, hire_date, salary, department_id,
job_id, manager_id
FROM employees
WHERE department_id = 40
ORDER BY departemnt_id;
```

Exemple2: Creació de la taula dept30 amb els camps id i name que corresponen amb identificador i nom dels empleats del departament 30.

```
CREATE TABLE dept30 (id, name) AS
SELECT employee_id, first_name
FROM employees
WHERE department_id = 30;
```

COPIA DE FILES DES D'UNA ALTRA TAULA

Mitjançant la sentència **INSERT INTO** podem inserir les files retornades per una consulta a una taula existent.

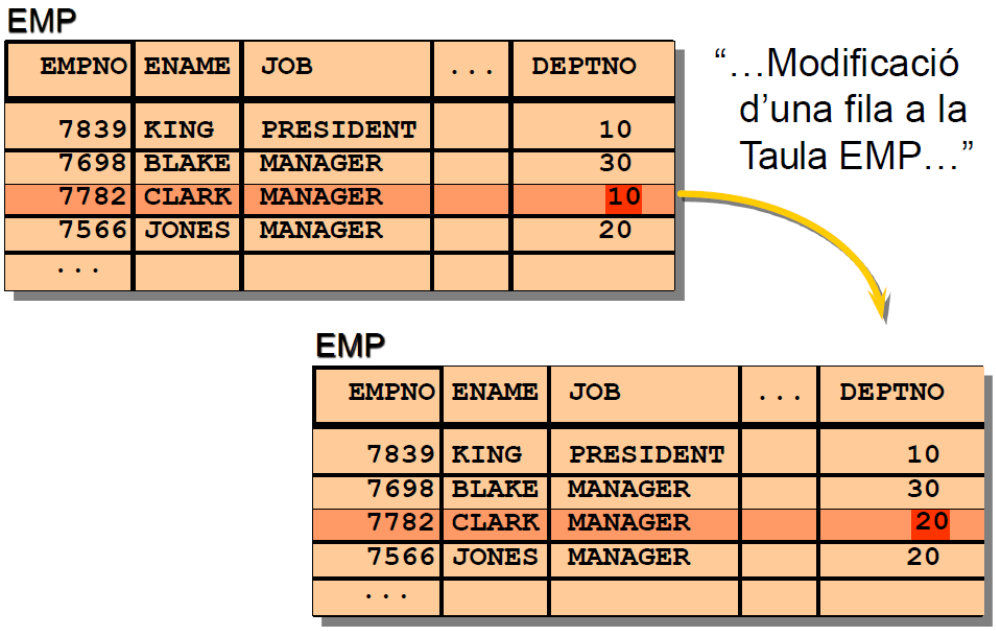
- Escriure la comanda INSERT amb una subconsulta (subquery)
- No hem d'utilitzar la clàusula VALUES
- Observar que coincideixi el nombre de columnes en la clàusula INSERT amb les de la subconsulta.

Exemple: Imaginem que hem creat una taula anomenada 'managers' amb l'identificador d'empleat, nom, salari i data de contractació. Inserim tots els empleats que tinguin com a feina 'MANAGER'.

```
INSERT INTO maanagers (id, name, salary, hire_date)
SELECT employee_id, first_name, salary, hire_date
```

```
FROM employees
WHERE job_id = 1;
```

CANVI DE LES DADES D'UNA TAULA



- Modifiquem registres existents amb la sentència **UPDATE**
- La modificació pot afectar més d'una fila alhora.

```
UPDATE     table
SET        column = value [, column = value]
[WHERE     condition]
[RETURNING *|column, column,…];
```

- Les files a modificar s'indiquen mitjançant la clàusula WHERE.

Exemple: Canviem el codi de departament de l'empleat amb id = 104.

```
UPDATE employees
SET department_id = 20
WHERE employee_id = 104;
```

[!Important]
Si s'omet la clàusula WHERE es modifiquen totes les files de la taula.
La següent consulta assigna a tots els empleats el departament 20.
UPDATE employees
SET department_id = 20;

Podem utilitzar subconsultes en la sentència UPDATE per modificar registres d'una taula, basats en valors d'una altra taula.

> Exemple: Update utilitzant subconsultes. Actualitzem el departament de tots els empleats que tenen la mateixa feina que l'empleat 102 amb el departament de l'empleat 101.

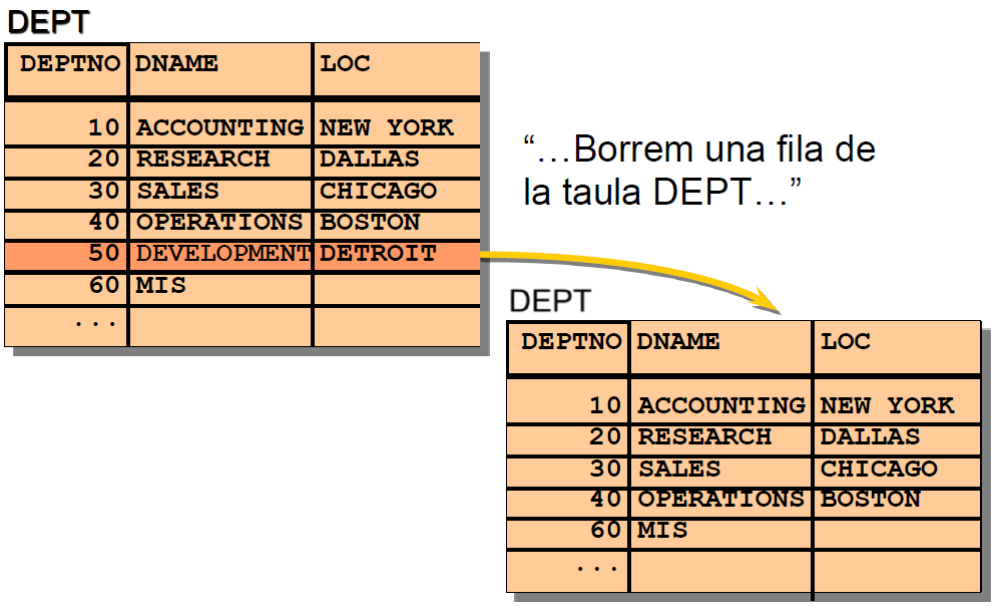
```
UPDATE employees
SET department_id = (SELECT department_id
                     FROM employees
                     WHERE employee_id = 101)
WHERE job_id = (SELECT job_id
                FROM employees
                WHERE employee_id = 102);
```

Es poden utilitzar també la condició de 'JOIN' per actualitzar les dades d'una taula a partir de les dades d'una altra.

> Exemple: Actualitzem el salari dels treballadors del departament 10 amb el salari mínim de la feina que tenen assignada incrementat en un 10%

```
UPDATE employees e
SET e.salary = j.min_salary * 1.10
FROM jobs j
WHERE e.job_id = j.job_id
AND e.department_id = 10;
```

ELIMINACIÓ DE FILES EN UNA TAULA



- Eliminem les files d'una taula mitjançant la sentència **DELETE**

```
DELETE FROM   table
[WHERE        condition]
[RETURNING *|column, column, …];
```

- Les files a eliminar s'indiquen mitjançant la clàusula WHERE.

> Exemple: Eliminem el departament 80
>
> ```
> DELETE FROM departments
> WHERE department_id = 80;
> ```

> [!Important]
> **Si s'omet la clàusula WHERE s'eliminen totes les files de la taula.**
> La següent consulta elimina tots els departaments.
> DELETE FROM departments;

Podem utilitzar subconsultes en la sentència DELETE per eliminar registres d'una taula, basats en valors d'una altra taula.

> Exemple: Delete utilitzant subconsultes. Eliminem els treballadors del departament anomenat 'Operations'.
>
> ```
> DELETE employees
> WHERE department_id = (SELECT department_id
>                        FROM departments
>                        WHERE department_name = 'Operations');
> ```

Es poden utilitzar també la condició de 'JOIN' per eliminar dades d'una taula a partir de les dades d'una altra.

> Exemple: Eliminen les files d'una taula 'contacts' a partir de les files existents a la taula 'blacklist'.
>
> ```
> DELETE FROM contacts
> USING blacklist
> WHERE contacts.phone = blacklist.phone;
> ```

ELIMINACIÓ DE LES DADES D'UNA TAULA AMB TRUNCATE TABLE

Mitjançant la sentència **TRUNCATE TABLE** considerada una sentència DDL, podem eliminar totes les files d'una taula, mantenint només l'estructura de la mateixa. Seria l'equivalent a fer un DELETE FROM sense especificar clàusula WHERE.

> Exemple: Eliminem totes les files de la taula 'managers'
>
> ```
> TRUNCATE TABLE managers;
> ```

CLÀUSULA RETURNING

Les sentències INSERT, UPDATE i DELETE, tenen opcionalment la clàusula RETURNING on podem indicar * o bé una llista de camps separats per coma.
Aquesta clàusula provoca que s'executi una sentència SELECT una vegada executada la sentència INSERT, UPDATE o DELETE que ens mostrarà les files inserides, actualitzades o eliminades.

TRANSACCIONS

Les transaccions són **l'element fonamental** de les bases de dades relacionals que ens permeten complir amb les propietats ACID explicades en el Tema 1. ç

Contenen una de les sentències següents:

- Conjunt de sentències DML que decideixen un canvi consistent sobre les dades. (Llenguatge de modificació de dades INSERT/UPDATE/DELETE)
- Una sentència DDL . (Llenguatge de definició de dades CREATE/DROP)
- Una sentència DCL (Llenguatge de control de les dades: GRANT, REVOKE)

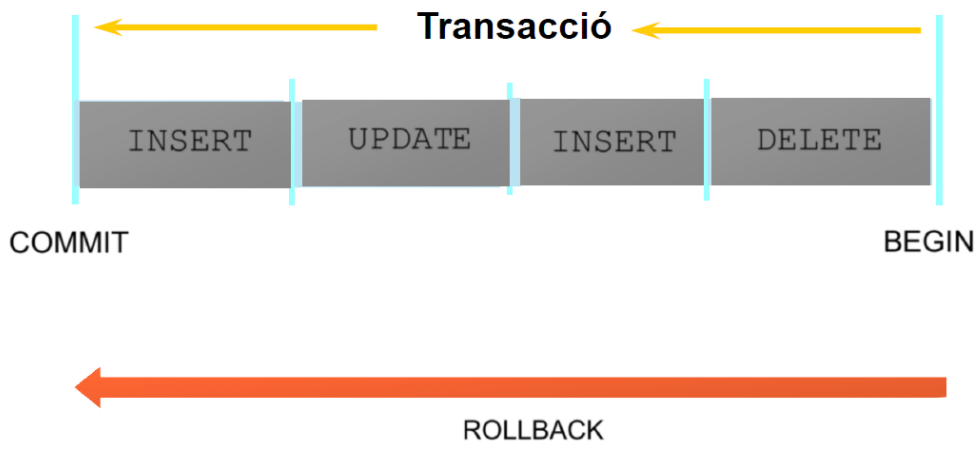
PostgreSQL utilitza l''AUTOCOMMIT' per a la validació de les sentències executades.
Des de psql:
\echo :AUTOCOMMIT
\set AUTOCOMMIT [on|off]
Amb AUTOCOMMIT OFF s'iniciarà una transacció implícita quan executem qualsevol sentència DML, DDL, DCL, haurem de tancar-la amb COMMIT o ROLLBACK.
Amb AUTOCOMMIT ON s'iniciarà una transacció implícita quan executem qualsevol sentència DML, DDL, DCL, aquesta transacció es tancarà després de l'execució de la sentència amb COMMIT si tot ha funcionat o ROLLBACK si hi ha algun error.

Per treballar amb transaccions i iniciar una nova transacció ho farem amb la sentència **BEGIN [WORK|TRANSACTION]**
Tancarem la transacció amb:

- **COMMIT [WORK|TRANSACTION]**, si volem confirmar els canvis.
- **ROLLBACK [WORK|TRANSACTION]**, si volem desfer els canvis.

Avantatges de les transaccions:

- Asseguren la consistència de les dades.
- Poden visualitzar els canvis sobre les dades abans de fer les permanents.
- Agrupen lògicament tasques relacionades entre si.



ESTAT DE LES DADES ABANS DE COMMIT/ROLLBACK

- Pot recuperar l'estat anterior de les dades.
- L'usuari actual pot revisar els resultats d'operacions DML usant la sentència SELECT.
- Altres usuaris no poden veure els resultats de les sentències DML executades per l'usuari actual, ja que s'executen en un entorn aïllat.
- Les files afectades són bloquejades, altres usuaris no poden canviar les dades pertanyents a aquestes files.

> Exemple: Operació d'actualització dins una transacció

```
BEGIN;

UPDATE employees
SET department_id = 10
WHERE department_id = 20;

COMMIT;
```

ESTAT DE LES DADES DESPRÉS D'UN COMMIT

- Els canvis en les dades són escrits a la base de dades.
- L'estat anterior de les dades es perd permanentment.
- Tots els usuaris poden veure els resultats.
- S'alliberen els bloquejos aplicats a les files afectades; aquestes files estan ara disponibles perquè altres usuaris les facin servir.

> Exemple: Operació d'esborrat dins una transacció que desfem.

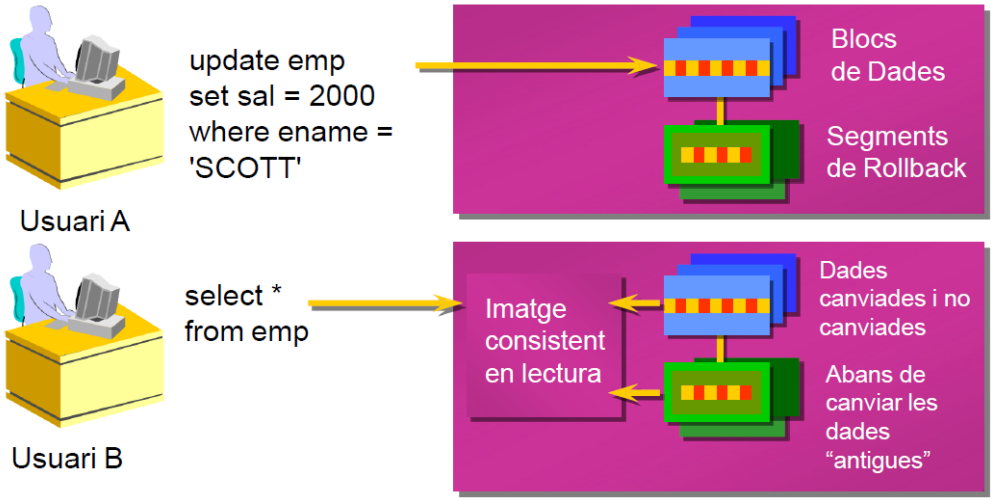
```
BEGIN;

DELETE FROM departments;

ROLLBACK;
```

ESTAT DE LES DADES DESPRÉS D'UN ROLLBACK

- Els canvis de dades es desfan.
- Es restaura l'estat anterior de les dades.
- S'alliberen tots els bloquejos sobre les files afectades.

CONSISTÈNCIA EN LA LECTURA DE DADES

- La consistència en lectura garanteix una vista consistent de les dades en qualsevol circumstància.
- Els canvis realitzats per un usuari no creen conflictes amb els canvis realitzats per un altre usuari.
- Assegura que sobre les mateixes dades les lectures no esperen a les escriptures i viceversa.



BLOQUEJOS

- Prevenen la interacció destructiva entre transaccions concurrents.
- No requereixen accions de l'usuari.
- Automàticament utilitzen el nivell més baix de restricció.
- Es mantenen mentre dura la transacció.

Els SGBD tenen bàsicament dues maneres de realitar bloquejos:

- Explícit
- Implícit

Bloqueig implícit: Realitzat per el SGBD segons l'operació que s'està executant.

- Dos tipus de bloqueig:
    - Exclusiu : Bloqueja altres usuaris
    - Compartit : Permet accés a altres usuaris
- Alt nivell de simultaneïtat de dades:
    - DML: Permet compartir la taula, és exclusiu de fila.
    - Consultes: No requereixen bloquejos.
    - DDL: Protegeix les definicions d'objectes.
    - Bloquejos mantinguts fins a COMMIT o ROLLBACK