
Complements d'SQL

PID_00253056

Alexandre Pereiras Magariños
M. Elena Rodríguez González

Temps mínim de dedicació recomanat: 9 hores





Els textos i imatges publicats en aquesta obra estan subjectes –llevat que s'indiqui el contrari– a una llicència de Reconeixement-NoComercial-SenseObraDerivada (BY-NC-ND) v.3.0 Espanya de Creative Commons. Podeu copiar-los, distribuir-los i transmetre'ls públicament sempre que en citeu l'autor i la font (FUOC. Fundació per a la Universitat Oberta de Catalunya), no en feu un ús comercial i no en feu obra derivada. La llicència completa es pot consultar a <http://creativecommons.org/licenses/by-nc-nd/3.0/es/legalcode.ca>

Índex

Introducció.....	5
Objectius.....	6
1. Claus subrogades.....	7
1.1. Concepte	7
1.2. Beneficis de les claus subrogades en el disseny de magatzems de dades	10
1.3. Construcció de claus subrogades a PostgreSQL	12
1.3.1. Seqüències	12
1.3.2. Tipus de dada <i>serial</i>	13
2. Common table expression.....	15
2.1. Concepte	15
2.2. Beneficis de l'ús de CTE	18
2.3. Construcció de CTE a PostgreSQL	18
2.3.1. Consultes recursives	19
2.3.2. Consultes CTE amb sentències de manipulació de dades	27
3. Funcions analítiques.....	29
3.1. Concepte	29
3.2. Beneficis de les funcions analítiques	35
3.3. Funcions analítiques a PostgreSQL	36
3.4. Tipus de funcions analítiques a PostgreSQL	41
3.4.1. <i>Row number</i>	41
3.4.2. <i>Rank</i>	43
3.4.3. <i>Dense rank</i>	44
3.4.4. <i>Lag</i>	45
3.4.5. <i>Lead</i>	47
3.4.6. <i>First value</i>	48
3.4.7. <i>Last value</i>	49
3.5. Ús de funcions d'agregació com a funcions analítiques	51
4. Tractament de valors nuls.....	54
4.1. Valors nuls en BD operacionals	54
4.2. Valors nuls en magatzems de dades	56
4.2.1. Valors nuls en taules de dimensions	56
4.2.2. Valors nuls en taules de fets	62
5. Transaccions.....	65

5.1.	Problemàtica associada a la gestió de transaccions	65
5.2.	Definició i propietats de les transaccions	68
5.3.	Interferències entre transaccions	69
5.4.	Nivell de concurrència	75
5.5.	Visió externa de les transaccions	76
5.5.1.	Relaxació del nivell d'isolament	79
5.5.2.	Responsabilitats de l'SGBD i del desenvolupador	81
5.6.	Transaccions a PostgreSQL	82
5.7.	Importància de les transaccions en OLTP enfront d'OLAP	84
Resum		86
Exercicis d'autoavaluació		87
Solucionari		90
Glossari		95
Bibliografia		97
Annexos: complements d'SQL – anotacions per a Oracle		98

Introducció

En aquest mòdul didàctic ampliarem els coneixements que tenim de l'SQL estàndard; més concretament, estudiarem un conjunt de funcionalitats que els sistemes de gestió de bases de dades (SGBD) implementen i que són molt útils per al desenvolupament d'aplicacions en entorns *data warehouse*. Concretament, treballarem amb els conceptes de claus subrogades, *common table expression* i funcions analítiques, conceptes que resulten de gran utilitat a l'hora d'implementar processos de càrrega de dades (ETL) i desenvolupament d'informes analítics.

A més, com a informació complementària, estudiarem la problemàtica de l'existència de valors nuls tant en bases de dades (BD) operacionals com en magatzems de dades, i com solucionar de manera eficient aquests problemes.

Finalment, veurem el concepte de transacció i les seves propietats, i estudiarem la problemàtica de l'accés simultani a les dades per part dels usuaris, i com els SGBD gestionen aquests escenaris de manera segura.

Cal tenir en compte que sovint hi ha diferències entre el que diu l'estàndard SQL (l'última versió del qual en el moment d'escriure aquest mòdul és SQL:2012) i les implementacions dels diversos proveïdors d'SGBD relacionals. En cadascuna de les seccions d'aquest mòdul es mostra, a manera d'exemple, com aplicar els conceptes explícits utilitzant l'SGBD relacional PostgreSQL. Com a informació complementària, es proporciona un annex en el qual es descriu la implementació d'aquests mateixos conceptes utilitzant en aquest cas l'SGBD relacional Oracle.

Les versions utilitzades d'aquests dos SGBD relacionals són PostgreSQL 9.3 i Oracle 11g respectivament, per la qual cosa en successives versions serà necessari consultar els manuals de cadascun dels fabricants per a identificar, si és el cas, els canvis respecte a la documentació presentada en aquest mòdul didàctic.

BD

Base de dades

SGBD

Sistema gestor de bases de dades

ETL

De l'anglès *extract, transform and load*, són el conjunt de processos en entorns *data warehouse* que s'encarreguen de l'extracció de dades procedents de múltiples orígens, de la transformació d'aquestes per a adequar-les a les noves estructures, i de la seva càrrega final en el magatzem de dades per al seu consum.

Magatzem de dades

En anglès *data warehouse* són bases de dades orientades a àrees d'interès de l'empresa que integren dades de diferents fonts amb informació històrica i no volàtil, i que tenen com a objectiu principal servir de suport en la presa de decisions.

Objectius

Aquest mòdul didàctic presenta conceptes avançats d'SQL per al tractament de dades. Una vegada finalitzat el seu estudi, tindreu les eines necessàries per a assolir els següents objectius:

1. Entendre el concepte de clau subrogada i els diferents mecanismes utilitzats per a la seva generació.
2. Conèixer, entendre i generar consultes *common table expression*.
3. Conèixer, entendre i generar consultes SQL recursives.
4. Conèixer, de manera general, les característiques i funcionalitat de les funcions analítiques, i saber aplicar-les per a l'obtenció de càlculs complexos.
5. Entendre la problemàtica dels valors nuls en bases de dades operacionals i entorns *data warehouse*, i aplicar les solucions apropiades.
6. Comprendre què és una transacció, les seves propietats i les funcions que ha de complir un SGBD en la gestió de transaccions.
7. Ser capaç de desenvolupar aplicacions que utilitzin correctament i amb eficiència els serveis de gestió de transaccions que ofereixen els SGBD.

1. Claus subrogades

En els SGBD, les claus primàries d'una taula poden definir-se a partir d'una sola columna, i es denomina **clau primària simple**, o bé com una combinació de columnes, cas en el qual s'anomena **clau primària composta**. Existeixen casos en els quals l'SGBD incorre en un cert nivell d'ineficiència a l'hora de fer operacions de combinació en consultes, ineficiència que s'observa quan el dissenyador es veu obligat a generar les claus primàries de les taules a partir d'una quantitat considerable de columnes (cosa que requereix que l'SGBD faci operacions de combinació sobre múltiples columnes), o fins i tot, sent una clau primària simple, quan els valors d'aquesta columna tenen prou grandària (assumint en aquest últim cas un tipus de dades alfanumèric). Aquesta ineficiència podria reduir-se o eliminar-se si el tipus de dada de la clau primària fos numèric i de menor grandària, i, per tant, fos una clau primària simple. Amb la finalitat de millorar aquesta problemàtica, es proposa el concepte de **claus subrogades**, concepte que veurem en les següents seccions.

1.1. Concepte

El concepte de claus subrogades és molt utilitzat dins d'implementacions de *data warehouse* o magatzem de dades i en entorns de *business intelligence* o intel·ligència de negoci. Per a entendre el concepte de **clau subrogada**, és necessari definir primer el concepte de **clau de negoci**.

Entenem per **clau de negoci** (en anglès, *business key*) el conjunt de columnes que conformen la clau primària d'una taula, que generalment té un significat propi d'acord amb les regles de negoci del sistema.

S'entén per **clau subrogada** (en anglès, *surrogate key*) l'identificador únic d'una taula que no es deriva de les dades de l'aplicació i que generalment no és visible a l'usuari. Sol construir-se a partir d'una seqüència numèrica autogenerada (amb valors enters, sense decimals) en què no existeix una relació entre el significat de la fila i aquesta clau. Una clau subrogada està sempre formada per una única columna.

La distinció entre clau de negoci i clau subrogada està molt present en **models multidimensionals** i processos ETL. Dins d'aquest context, la **clau subrogada** representa la **clau primària de la taula destí**, i la **clau de negoci** representa la **clau primària de la taula origen** (des de la qual es llegeixen les dades) i que també se sol emmagatzemar en la taula destí.

Models multidimensionals

Un model multidimensional és aquell que concep les dades que volem analitzar en termes de fets i dimensions d'anàlisi, de manera que els podem situar en un espai n-dimensional.

Claus subrogades

Les claus subrogades també reben el nom de claus sense significat (*meaningless keys*), claus enteres (*integer keys*), claus artificials (*artificial keys*), claus no naturals (*non-natural keys*) o claus substitutes (*substitute keys*).

Exemple de claus de negoci i claus subrogades en un model multidimensional

Suposem que tenim la taula Assignatura amb quatre columnes: Id, Codi, Nom de l'assignatura i Data d'alta. En aquest exemple, Id és clau primària i clau subrogada (no té un significat apparent, és una seqüència numèrica), i Codi és la clau de negoci (amb un significat especial dins del sistema origen).

Assignatura			
<u>Id</u>	Codi	Nom	Data d'Alta
1	UOC-12345	Bases de dades	10/01/2000
2	UOC-66521	Enginyeria del programari	10/01/2000
3	UOC-98321	Programació	10/01/2000
4	UOC-21473	Àlgebra	10/01/2000

El fet de mantenir aquesta separació en models multidimensionals facilita el control dels canvis que els sistemes origen puguin patir en el cas que aquests reutilitzin els valors de la clau de negoci al llarg del temps. D'aquesta manera, preservem l'històric de canvis per a facilitar l'anàlisi de les dades.

Exemple de reutilització de claus de negoci

És freqüent que les bases de dades operacionals reutilitzin valors de clau primària després d'un període d'inactivitat. Un cas concret seria el dels números de telèfon mòbil.

Suposem que tenim una base de dades amb una taula de números de telèfon d'una companyia de telecomunicacions, en la qual es guarda el número de telèfon i el nom del titular. La clau primària d'aquesta taula és el número de telèfon.

Base de dades operacional

Base de dades destinada a gestionar el dia a dia d'una organització, és a dir, emmagatzema la informació referent a la gestió operativa diària d'una institució.

Números de telèfon	
<u>Número Telèfon</u>	Nom Titular
655138007	Manuela Domínguez
655138006	José López
655138005	Juan Manuel Carrillo

En el cas en què un client doni de baixa un número de telèfon, el número podria ser assignat de nou a un altre client després d'un període d'inactivitat (per exemple, dotze mesos). Aquest seria el cas del número 655138007, que, com es pot veure en la següent taula, ha canviat el titular a José Sánchez, a diferència del titular que prèviament tenia assignat aquest número de telèfon (Manuela Domínguez).

Números de telèfon	
<u>Número Telèfon</u>	Nom Titular
655138007	José Sánchez
655138006	José López
655138005	Juan Manuel Carrillo

La problemàtica és que, en aquest escenari, no mantenim un històric dels titulars. Aquest escenari es pot representar molt bé en models multidimensionals.

Vegeu ara el següent exemple, en què tenim una dimensió de números de telèfon (que utilitza la taula de la base de dades operacional com a origen de dades) amb una clau subrogada com a clau primària. Veiem que el número de telèfon 655138007 pot reutilitzar-se i mantenir el titular original, i que se separa així la clau de negoci de la clau subrogada. D'aquesta manera podríem, per exemple, assignar correctament les trucades fetes per cada client.

Dimensió Números de Telèfon		
<u>Id</u>	Número Telèfon	Nom Titular
1	655138007	Manuela Domínguez
2	655138006	José López
3	655138005	Juan Manuel Carrillo
4	655138007	José Sánchez

Magatzem de dades

Els magatzems de dades (en anglès, *data warehouse*) són BD orientades a àrees d'interès de l'empresa que integren dades de diferents fonts amb informació històrica i no volàtil, i que tenen com a objectiu principal servir de suport en la premsa de decisions.

Hi ha un cas concret en el qual la clau subrogada podria tenir un significat especial. A l'hora de dissenyar un magatzem de dades, hi ha una taula molt important denominada **Data** o **Dia**, que representa els dies del calendari, és a dir, cada fila emmagatzemarà una data concreta: 1 gener 2015, 15 juliol 2002, etc. i les característiques associades a cada data (dia de la setmana, si és cap de setmana, si és festiu...). En el cas especial d'aquesta taula, la clau subrogada se sol representar com un enter el valor del qual és la data representada en format YYYYMMDD. Utilitzant els exemples de dates anteriors, l'1 de gener de 2015 es representaria com a 20150101, i el 15 de juliol de 2002 es representaria com a 20020715.

Exemple de dimensió Data

Vegeu l'exemple proposat per a una dimensió Data, on la clau primària (i subrogada) és la data en format YYYYMMDD.

Dimensió Data						
<u>Id_Data</u>	Data	Dia de la Setmana	Mes	Mes (Dígits)	Trimestre	Any
20150101	01/01/2015	Dijous	Gener	1	Q1	2015
20150102	02/01/2015	Divendres	Gener	1	Q1	2015
20150103	03/01/2015	Dissabte	Gener	1	Q1	2015
20150104	04/01/2015	Diumenge	Gener	1	Q1	2015

Una de les raons principals per la qual la clau subrogada de la dimensió Data té un significat és, a diferència d'altres, facilitar una partició física de les dades eficient: la possibilitat de separar dades físicament sobre la base del valor de la columna Data permet no solament millorar el rendiment de la consulta a

l'hora de consultar les dades històricament, sinó també el seu manteniment, cosa que facilita la inserció de noves dades i purgar dades històriques sense que aquestes dues operacions s'affectin mútuament.

1.2. Beneficis de les claus subrogades en el disseny de magatzems de dades

L'ús de claus subrogades en l'àmbit dels magatzems de dades, a més de ser una bona pràctica a l'hora de dissenyar-los, comporta una sèrie de beneficis.

Com ja s'ha esmentat anteriorment, un dels principals avantatges d'utilitzar claus subrogades és crear una separació en models multidimensionals per a facilitar el control dels canvis en els sistemes origen a l'hora de reutilitzar valors de la clau de negoci (vegeu l'exemple dels números de telèfon mostrat anteriorment).

Un altre avantatge molt important és la millora de rendiment en operacions de consulta. Les operacions de combinació (`JOIN`) entre les diferents taules dins d'un model multidimensional (dimensions i fets) es faran mitjançant aquestes claus subrogades, el tipus de dades de les quals és, com ja hem esmentat, numèric-enter, i comparades amb altres tipus de dades, com dates o cadenes de caràcters alfanumèriques, soLEN ocupar menys espai. El fet que ocupin menys espai significa que les taules de fets, que contenen les claus foranes de les dimensions, siguin més petites, i per tant els seus índexs seran també més petits. Això es tradueix en una reducció del nombre de pàgines utilitzades per a emmagatzemar files (podem emmagatzemar més files per pàgina), per la qual cosa podrem llegir més dades amb menys operacions d'entrada/sortida (E/S).

Un altre dels beneficis importants de l'ús de claus subrogades és el de facilitar el **seguiment de canvis sorgits en les BD operacionals** per a facilitar l'anàlisi de dades i mantenir l'històric de valors i descripcions (al contrari que en un sistema operacional, tal com hem vist en l'exemple de reutilització de números de telèfon).

Exemple de seguiment de canvis en un model multidimensional

Suposem que, en l'exemple de la taula Assignatura, l'assignatura amb codi UOC-21473 (Àlgebra) ja no s'imparteix més i el seu codi és reutilitzat per a l'assignatura Programació orientada a objectes. En aquest escenari, s'afegiria una nova fila amb el mateix codi, i el nou nom i data d'alta.

Referència bibliogràfica

Per a veure les diferents tècniques de seguiment de canvis en un magatzem de dades, es recomana revisar la següent referència bibliogràfica:

Kimball, R.; Ross, M. (2013). *The Data Warehouse Toolkit* (3a. ed.). John Wiley & Sons, Inc.

Assignatura			
<u>Id</u>	Codi	Nom	Data Alta
1	UOC-12345	Bases de dades	10/01/2000
2	UOC-66521	Enginyeria del programari	10/01/2000
3	UOC-98321	Programació	10/01/2000
4	UOC-21473	Àlgebra	10/01/2000
5	UOC-21473	Programació orientada a Objectes	15/09/2015

Les claus subrogades faciliten també la **integració i consolidació de dades des de múltiples orígens de dades**, tot i que la **clau de negoci** en els diferents sistemes **no sigui homogènia** en tots.

Exemple d'ús de claus subrogades per a la integració de dades entre múltiples orígens

Suposem que tenim dos sistemes que emmagatzemen informació d'usuari. Els usuaris en cada sistema es codifiquen de manera diferent: el primer sistema utilitza noms d'usuari en format UXxxxx, on XXXXX és una seqüència numèrica, mentre que el segon sistema utilitza un format ADDXXXX, on DD és el codi de departament i XXXX és una seqüència numèrica. Aquest escenari es podria representar en un model multidimensional amb claus subrogades de la següent manera:

Usuari		
<u>Id</u>	Id usuari	Sistema origen
1	U00001	SISTEMA 1
2	U00002	SISTEMA 1
3	AHR0001	SISTEMA 2
4	AHR0002	SISTEMA 2
5	U00003	SISTEMA 1

L'ús de claus subrogades també ens permet codificar eficientment la **falta de valors en una base de dades operacional**. D'aquesta manera, podem assignar un valor de clau subrogada a un valor **per defecte** i mapar-lo correctament a la taula de fets.

Exemple de valors per defecte

És comú que hi hagi una fila amb una clau subrogada especial per a mapar la inexistència de valors en la base de dades operacional. En l'exemple d'Assignatura, s'ha afegit una fila amb clau subrogada -1 que s'utilitzarà per a identificar la falta de codis d'assignatures:

Assignatura		
<u>Id</u>	Codi	Nom
-1	Desconeuguda	Desconeuguda
1	UOC-12345	Bases de dades
2	UOC-66521	Enginyeria del programari
3	UOC-98321	Programació
4	UOC-21473	Àlgebra

1.3. Construcció de claus subrogades a PostgreSQL

En aquesta secció veurem com es poden construir claus subrogades utilitzant l'SGBD PostgreSQL. Per a això, suposem que disposem de la següent definició de la taula Assignatura: `assignatura_key` (clau subrogada i clau primària), `cod_assignatura` (codi d'assignatura) i `nom_assignatura` (nom de l'assignatura). A continuació podem veure la definició d'aquesta taula en llenyatge SQL.

```
CREATE TABLE assignatura (
    assignatura_key INTEGER PRIMARY KEY,
    cod_assignatura CHARACTER VARYING(10) NOT NULL,
    nom_assignatura CHARACTER VARYING(100) NOT NULL
)
```

Notació

La notació per a representar la sintaxi de les sentències SQL serà la següent:

- Les paraules en majúscules són paraules reservades del llenguatge.
- Les paraules en minúscules són noms d'estructures de la BD creades per l'usuari (taules, columnes, etc.).
- La notació [...] vol dir que el que hi ha entre els claudàtors és opcional.
- La notació {A|...|B} vol dir que hem d'escollar entre totes les opcions que hi ha entre les claus, però que hem de posar-ne una obligatòriament.

La generació de claus subrogades es pot implementar de les següents maneres: utilitzant seqüències i utilitzant el tipus `serial`.

1.3.1. Seqüències

La creació d'una seqüència ens permet generar valors numèrics de manera consecutiva. La definició de seqüències en PostgreSQL és la següent:

```
CREATE [ TEMPORARY | TEMP ] SEQUENCE name [ 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 } ]
```

Com a exemple, a continuació definirem la seqüència que utilitzarà la taula Assignatura:

```
CREATE SEQUENCE seq_assignatura_key INCREMENT BY 1 START WITH 1 NO CYCLE;
```

Per a fer ús de la seqüència i així generar nous valors, disposem de la funció `nextval(seq_name)`, on `seq_name` representa el nom de la seqüència a utilitzar. Aquesta funció s'encarrega d'avançar al següent valor de la seqüència i retornar aquest valor. Aquestes dues operacions es fan de manera atòmica, per la qual cosa múltiples sessions poden fer ús de la mateixa seqüència sense preocupar-se que els valors es reutilitzin.

```
SELECT nextval('seq_assignatura_key')
```

A l'hora d'inserir una fila nova a la taula, podem fer una crida a la funció com a part de la sentència `INSERT`:

```
INSERT
INTO
    assignatura
(
    assignatura_key,
    cod_assignatura,
    nom_assignatura
)
VALUES
(
    nextval('seq_assignatura_key'),
    'UOC-35298',
    'Arquitectura de Dades'
)
```

Nota

L'explicació de les opcions de les diferents clàusules utilitzades en la creació d'una seqüència es pot consultar en la següent URL de PostgreSQL:
<http://www.postgresql.org/docs/9.3/static/sql-createsequence.html>

Exercici

Investiga l'ús de les funcions de manipulació de seqüències en la següent URL:
<http://www.postgresql.org/docs/9.3/static/functions-sequence.html>

1.3.2. Tipus de dada `serial`

Una altra manera de definir claus subrogades és mitjançant la creació d'una columna amb tipus de dades `serial` (o bé `smallserial` i `bigserial`, dependent de la capacitat numèrica que es necessiti per a emmagatzemar els valors).

Si decidim utilitzar aquest mecanisme, la taula Assignatura s'haurà de definir de la següent manera (fixeu-vos com la definició de la columna `assignatura_key` ha canviat pel que fa a la definició original):

```
CREATE TABLE assignatura (
    assignatura_key SERIAL PRIMARY KEY,
    cod_assignatura CHARACTER VARYING(10) NOT NULL,
    nom_assignatura CHARACTER VARYING(100) NOT NULL
)
```

Cada vegada que s'insereix una nova fila a la taula, la columna assignatura_key generarà un nou valor. En els següents exemples es pot veure que es pot especificar el valor amb la clàusula DEFAULT o, simplement, no incloure aquesta columna com a part de la sentència INSERT:

```
INSERT INTO assignatura (assignatura_key, cod_assignatura, nom_assignatura)
VALUES (DEFAULT, 'UOC-35299', 'Sistemes Operatius');

INSERT INTO assignatura (cod_assignatura, nom_assignatura)
VALUES ('UOC-35298', 'Arquitectura de Dades');
```

En realitat, els tipus de dades serial i les seves variants no són tipus de dades reals. Aquestes no són més que una notació especial a PostgreSQL que es tradueix en la creació d'una seqüència i l'assignació d'un valor per defecte a la columna en qüestió. Utilitzant l'exemple anterior, el resultat de crear aquesta taula amb aquest tipus de dada seria equivalent a especificar les següents estructures:

```
CREATE SEQUENCE seq_assignatura_key;
CREATE TABLE assignatura (
    assignatura_key INTEGER NOT NULL DEFAULT nextval('seq_assignatura_key') PRIMARY KEY,
    cod_assignatura CHARACTER VARYING(10) NOT NULL,
    nom_assignatura CHARACTER VARYING(100) NOT NULL
);
ALTER SEQUENCE seq_assignatura_key
    OWNED BY assignatura.assignatura_key;
```

2. Common table expression

Quan generem consultes SQL, es produueixen situacions en les quals necessitem fer operacions o càlculs sobre un conjunt de dades que no existeixen en el sistema de manera inherent, sinó que aquestes han de ser obtingudes mitjançant agregacions, combinacions entre taules, filters i càlculs sobre les dades existents.

Una manera d'obtenir aquest conjunt de dades és utilitzant vistes, amb les quals aconseguim dividir aquelles consultes complexes en parts més senzilles, encapsulant així el codi que necessitem per a obtenir el subconjunt de dades de les taules mestres que necessitem. El problema de les vistes és que són objectes permanents en el sistema, la qual cosa pot resultar un inconvenient a l'hora de crear consultes no planificades. Per exemple, en un entorn de producció (entorns que soLEN ser estrictes i restringits a l'hora de crear objectes en la BD). Una altra solució que els SGBD ens proporcionen és la de generar suconsultes. Però aquestes tenen el problema que dificulten la lectura del codi i el seu manteniment, i ens obliga, en certes situacions, a repetir el mateix codi en més d'una ocasió, a causa de la impossibilitat de referenciar una subconsulta al llarg de la consulta principal. Per a solucionar aquesta problemàtica, s'han introduït les *common table expression*, que ens ofereixen una manera més senzilla i elegant de generar consultes que requereixen dades no inherents en el sistema i que aquestes puguin referenciar-se de manera senzilla.

A continuació, veurem amb més detall què són les *common table expression* i quins són els beneficis d'aquest tipus de construccions, a més de proporcionar les clàusules a PostgreSQL necessàries per a la generació d'aquest tipus de consultes, clàusules que han estat afegides com a part de l'estàndard SQL:1999.

2.1. Concepte

Les CTE (de l'anglès *common table expression*) són una funcionalitat que proporciona SQL per a simplificar i facilitar la construcció de consultes complexes. Les CTE es creen a partir de la clàusula WITH.

Les CTE, mitjançant l'ús de la clàusula WITH, ens permeten definir consultes auxiliars per a utilitzar-les en consultes més complexes amb una única declaració. Aquestes consultes auxiliars permeten «trencar» aquesta consulta complexa en consultes més petites i llegibles, a més de permetre la reutilització d'aquestes petites consultes en més d'una ocasió dins d'una mateixa consulta. Podríem dir que aquestes consultes auxiliars tenen un comportament similar a

la construcció de taules temporals, ja que es tracta de consultes les dades resultants de les quals, d'alguna manera, són guardades temporalment per l'SGBD i es descarten una vegada la consulta ha finalitzat la seva execució.

Exemple de consulta CTE

Suposem que tenim la següent taula d'empleats. La columna Id Supervisor és l'identificador d'empleat que actua com a supervisor. En el cas que l'empleat no tingui un supervisor assignat, això significa que l'empleat és el director general de l'empresa.

Empleats				
Id Empleat	Nom	Ciutat	Salari Anual	Id Supervisor
1	Manuel Vázquez	Barcelona	23500	7
2	Elena Rodríguez	Tarragona	16000	1
3	José Pérez	Girona	17000	1
4	Alejandra Martínez	Barcelona	22500	7
5	Marina Rodríguez	Vilanova	12000	4
6	Fernando Nadal	Viladecans	13000	4
7	Victoria Suárez	Tarragona	31000	10
8	Victor Anllada	Lleida	28000	10
9	José María Llopis	Barcelona	29000	10
10	Victoria Setan	Castelldefels	45000	
11	Manuel Bertrán	Barcelona	21000	9

Donat aquest conjunt de dades, volem obtenir una llista de tots els empleats (nom i ciutat), la diferència entre el salari màxim i el salari de l'empleat, la diferència entre el salari mínim i el salari de l'empleat, i la diferència entre el salari mitjà i el salari de l'empleat, ordenat per l'identificador d'empleat ascendentment. El càlcul dels salaris màxim, mínim i mitjà ha de fer-se excloent el director general. Aquesta consulta es podria implementar de la següent manera:

```

SELECT
    e1.nom,
    e1.ciutat,
    e1.salari - (SELECT MAX(e2.salari)
                  FROM empleat e2
                  WHERE e2.id_supervisor IS NOT NULL) AS diferencia_max,
    e1.salari - (SELECT MIN(e3.salari)
                  FROM empleat e3
                  WHERE e3.id_supervisor IS NOT NULL) AS diferencia_min,
    e1.salari - (SELECT AVG(e4.salari)
                  FROM empleat e4
                  WHERE e4.id_supervisor IS NOT NULL) AS diferencia_avg
FROM
    empleat e1
ORDER BY
    e1.id_empleat
  
```

Com podem veure, el codi per a obtenir els salaris màxim, mínim i mitjà és molt similar: les tres subconsultes utilitzen la mateixa taula i la mateixa condició en la clàusula WHERE. En canvi, la consulta repeteix el mateix codi diverses vegades (taula i condicions imposades) en diferents parts. Què passaria si ara haguéssim d'eliminar la condició de

no incloure el director general? Què passaria si en lloc d'excloure el director general haguéssim d'excloure també els empleats de Barcelona? En aquests casos, hauríem de modificar el codi en tres llocs diferents, la qual cosa seria en certa manera ineficient i difícil de mantenir.

Utilitzant una consulta CTE construiríem la consulta de la següent manera. Vegeu que tota la lògica de la consulta està escrita en un lloc en concret mitjançant la clàusula WITH (com si fos una taula temporal denominada `salaris`), i que aquesta és després referenciada per a calcular les diferències. En el cas que necessitem modificar els criteris de selecció dels salaris, n'hi hauria prou de modificar la definició de la consulta auxiliar anomenada `salaris`. D'aquesta manera, se simplifica el codi i es facilita tant la lectura d'aquest com el seu manteniment, a més de millorar el rendiment de la consulta a causa que `salaris` s'avalua una única vegada.

```
WITH salaris AS (
    SELECT
        MAX(salari) AS max_salari,
        MIN(salari) AS min_salari,
        AVG(salari) AS avg_salari
    FROM
        empleat
    WHERE
        id_supervisor IS NOT NULL
)
SELECT
    nom,
    ciutat,
    salari - (SELECT max_salari FROM salaris) AS diferencia_max,
    salari - (SELECT min_salari FROM salaris) AS diferencia_min,
    salari - (SELECT avg_salari FROM salaris) AS diferencia_avg
FROM
    empleat
ORDER BY
    id_empleat
```

Els resultats d'ambdues consultes, que són equivalents, es mostren a continuació:

Nom	Ciutat	Diferència Màx	Diferència Mín	Diferència Avg
Manuel Vázquez	Barcelona	-7500	11500	2200
Elena Rodríguez	Tarragona	-15000	4000	-5300
José Pérez	Girona	-14000	5000	-4300
Alejandra Martínez	Barcelona	-8500	10500	1200
Marina Rodríguez	Vilanova	-19000	0	-9300
Fernando Nadal	Viladecans	-18000	1000	-8300
Victoria Suárez	Tarragona	0	19000	9700
Victor Anllada	Lleida	-3000	16000	6700
José María Llopis	Barcelona	-2000	17000	7700
Victoria Setan	Castelldefels	14000	33000	23700
Manuel Bertrán	Barcelona	-10000	9000	-300

2.2. Beneficis de l'ús de CTE

Entre els possibles beneficis de l'ús de *common table expression*, podem destacar-ne els següents:

- 1) Facilitar la llegibilitat i manteniment del codi: aquesta és una característica molt important, ja que ens permet definir càlculs complexos una única vegada i reutilitzar-los en diferents punts d'una mateixa consulta.
- 2) Generar i reutilitzar codis més eficientment: una de les propietats de les CTE és que s'avaluen una sola vegada per execució, per la qual cosa si la consulta principal ha d'utilitzar un càlcul complex en més d'una ocasió, guanyem en eficiència en l'execució de les consultes.

- 3) Construir consultes recursives: aquesta característica és de les més importants, ja que ens permet utilitzar SQL, entre altres coses, per a construir jerarquies de dades i, en general, resoldre problemes complexos que requereixin recursivitat.

2.3. Construcció de CTE a PostgreSQL

La possibilitat de crear consultes CTE a PostgreSQL s'ha introduït en la versió PostgreSQL 8.4, i la construcció d'aquestes es fa de la següent manera:

```
WITH [RECURSIVE] alias_1 [ ( column_1, column_2, ... ) ] AS (
    query_1
), [RECURSIVE] alias_2 [ ( column_1, column_2, ... ) ] AS (
    query_2
),
...
[RECURSIVE] alias_n [ ( column_1, column_2, ... ) ] AS (
    query_n
)
main_query
```

Primer es declaren les **consultes auxiliars** que volem definir utilitzant la clàusula `WITH`, cadascuna d'aquestes amb un àlies específic. Podrem definir tantes consultes auxiliars com siguin necessàries, i aquestes poden referenciar-se les unes a les altres, sempre que la consulta referenciada estigui declarada abans que la consulta que referencia. És a dir, la consulta definida com a `alias_n` només pot fer referència a les consultes definides anteriorment (des d'`alias_1` a `alias_{n-1}`). Finalment, es defineix el que denominem **consulta principal** (`main_query`).

En PostgreSQL, és important destacar que les consultes definides en el `WITH` només s'executarán si són referenciades a la consulta principal. A més, tant les consultes auxiliars definides com a part de la clàusula `WITH` com la consulta principal (`main_query`) poden ser tant expressions `SELECT` com expressions DML (`INSERT`, `UPDATE` o `DELETE`).

CTE Bounties

En la wiki de PostgreSQL CTE Bounties es presenten diversos exemples de quin tipus de problemes podem resoldre utilitzant CTE i la solució proposada. L'enllaç a aquest wiki és el següent:

https://wiki.postgresql.org/wiki/cte_bounties

Consulta recursiva

Una consulta recursiva és la consulta que permet referenciar-se a si mateixa, i s'implementa amb la tècnica de CTE. Les veurem en la següent secció.

DML

Recordeu que DML significa *data manipulation language*.

2.3.1. Consultes recursives

La clàusula opcional `RECURSIVE` s'utilitza per a indicar a PostgreSQL que es tracta d'una consulta recursiva, és a dir, que la consulta definida mitjançant la clàusula `WITH` pot referenciar-se a si mateixa.

Les consultes recursives a PostgreSQL, en la seva forma més simple, s'implementen de la següent manera:

```
WITH RECURSIVE aliases [ ( column_1, column_2, ... ) ] AS (
    non_recursive_term
    [ UNION | UNION ALL ]
    recursive_term
)
SELECT expression FROM aliases
```

Les parts que la formen són:

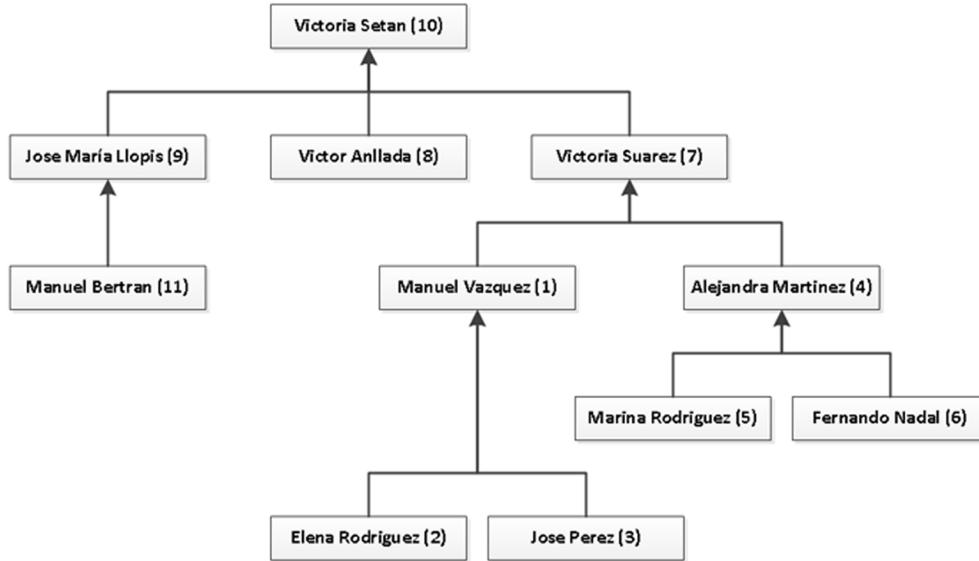
- 1) Declaració de la consulta recursiva mitjançant la clàusula `RECURSIVE`, un àlies i, opcionalment, una llista de columnes que es correspondran amb la llista de columnes resultat de la consulta.
- 2) Declaració de la part no recursiva (`non_recursive_term`), que mai no podrà referenciar la consulta recursiva.
- 3) `UNION` o `UNION ALL`, depenent de les necessitats: amb `UNION` eliminem els duplicats, amb `UNION ALL` els mantenim.
- 4) Declaració de la part recursiva (`recursive_term`), que sí que podrà referenciar a la consulta recursiva, i és la que ens permet fer la recursió (iteració).

Per a entendre la declaració i funcionament d'aquestes consultes, vegem el següent exemple.

Exemple de consulta recursiva: jerarquia d'empleats

Suposem que volem obtenir, per a cadascun dels empleats, l'estructura jeràrquica a l'empresa des del director general fins a l'empleat esmentat; això és, el nom del director general, el nom del subordinat, el nom del següent subordinat, etc. i així fins a arribar a l'empleat en qüestió. Per a facilitar l'exemple, es mostra en la figura següent la jerarquia de les dades d'empleat de què disposem en l'exemple.

Figura 1. Jerarquia d'empleats



Volem que el resultat de la consulta ens retorni, per a cada empleat, una columna amb el següent format:

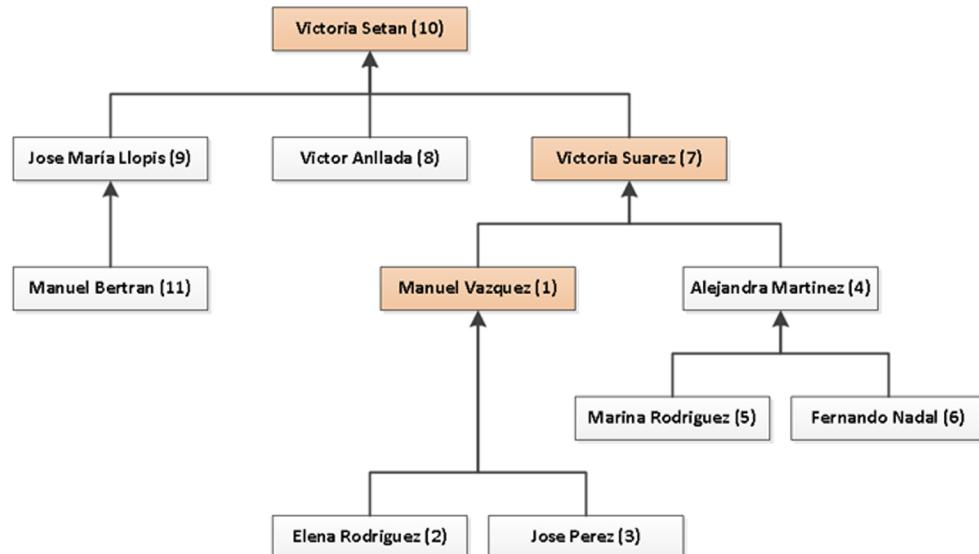
```
Director general <- Empleat nivell 1 <- Empleat nivell 2 <- ...
```

Concretament, i utilitzant com a exemple l'empleat Manuel Vázquez (amb identificador 1), la jerarquia associada seria la següent:

```
Victoria Setan <- Victoria Suárez <- Manuel Vázquez
```

En la figura 2, podem veure on es troba aquest empleat i quina és la jerarquia que porta associada (cel·les ressaltades):

Figura 2. Jerarquia de supervisors per a Manuel Vázquez



Fer aquesta consulta amb SQL i sense utilitzar CTE significaria fer dues operacions de combinació (`JOIN`) sobre la taula d'empleats: la primera per a obtenir el superior de Manuel Vázquez (que és Victoria Suárez), i la segona per a obtenir el superior de Victoria Suárez (que és Victoria Setan, directora general). La consulta s'implementaria de la següent manera:

```

SELECT
    e3.nom || ' <- ' || e2.nom || ' <- ' || e1.nom
FROM
    empleat e1
    INNER JOIN empleat e2
        ON e1.id_supervisor = e2.id_empleat
    INNER JOIN empleat e3
        ON e2.id_supervisor = e3.id_empleat
WHERE
    e1.nom = 'Manuel Vázquez'

```

Com podeu veure, això és molt poc eficient. En el cas que la jerarquia tingués més de tres nivells, la consulta hauria de ser modificada per a afegir més operacions de combinació. A més, en aquest cas concret sabem per endavant els nivells que hi ha entre Manuel Vázquez i el director general, que són tres. Què passaria si no disposéssim d'aquesta informació? Hauríem d'anar provant consultes fins a arribar al punt en el qual, pujant en la jerarquia, obtenim un empleat amb un identificador de supervisor `NULL`, que identifica el director general. Encara més: què hauríem de fer si ens demanessin aquestes mateixes dades per a tots els empleats de l'empresa? La solució utilitzant el mecanisme anterior seria inviable.

La solució per a aquest tipus de consultes és l'ús de consultes recursives. Suposant que haguéssim de presentar la jerarquia per a tots els empleats de l'empresa, la consulta que ens proporciona els resultats desitjats seria la que es mostra a continuació:

```

WITH RECURSIVE jerarquies AS (
    SELECT
        id_empleat,
        nom,
        id_supervisor,
        CAST (nom AS TEXT) AS resultat
    FROM
        empleat
    WHERE
        id_supervisor IS NULL
    UNION ALL
    SELECT
        e.id_empleat,
        e.nom,
        e.id_supervisor,
        CAST (j.resultat || ' <- ' || e.nom AS TEXT) AS resultat
    FROM
        empleat e INNER JOIN jerarquies j
            ON ( e.id_supervisor = j.id_empleat )
)
SELECT
    nom,
    resultat
FROM
    jerarquies
ORDER BY
    resultat

```

Destaquem els següents aspectes:

- Es pot veure que la part no recursiva declara una condició `id_supervisor IS NULL`. Aquesta és la condició que ens permet identificar l'últim empleat de la jerarquia (el director general).
- S'ha utilitzat en aquest cas `UNION ALL`. La raó principal és evitar l'eliminació de duplicats (encara que és força improbable que dues persones amb el mateix nom comparteixin la mateixa jerarquia).
- És important destacar que la funció `CAST` s'utilitza per a poder emmagatzemar, en una columna denominada `resultat`, totes les dades de la jerarquia. El `CAST` es fa en un tipus de dada `TEXT`, ja que desconeixem la longitud màxima en nombre de caràcters que el format desitjat ens ocuparà. En el cas que no es faci, i que el valor d'aquesta columna excedeixi el de la longitud màxima de la columna `nom` (que és la que s'utilitza per a obtenir la jerarquia, de cent caràcters), ens donarà un error:

```

ERROR: recursive query "jerarquies" column 4 has type character varying(100) in non-
recursive term but type character varying overall
LINE 3: (SELECT id_empleat, nom, id_supervisor, nom As resultat
          ^
HINT: Cast the output of the non-recursive term to the correct type.

***** Error *****

ERROR: recursive query "jerarquies" column 4 has type character varying(100) in non-
recursive term but type character varying overall
SQL state: 42804
HINT: Cast the output of the non-recursive term to the correct type.
Character: 72

```

Avaluació de consultes recursives

Com avaluava PostgreSQL les consultes recursives? Aquests són els passos que aquest SGBD segueix:

- Primer, s'avalua la part no recursiva (`non_recursive_term`). Aquestes dades s'emmagatzem en dos llocs: d'una banda, en l'espai destinat als resultats de la consulta `WITH`, que denominarem **taula intermèdia**, i, d'altra banda, en una taula denominada **taula de treball** (*working table* en anglès).
- A continuació, es fan els següents passos de manera iterativa:
 1. S'avalua la part recursiva (`recursive_term`), substituint la crida a la consulta recursiva per la taula de treball.
 2. Als resultats d'avaluar la part recursiva se'ls aplica `UNION` o `UNION ALL` amb les dades existents a la taula intermèdia, segons s'hagi especificat a la consulta. En el cas d'utilitzar `UNION`, s'eliminaran els duplicats.
 3. Finalment, aquests resultats obtinguts s'emmagatzemen a la taula de treball, havent eliminat prèviament els resultats de l'avaluació anterior.

El procés iteratiu s'acaba quan l'execució del pas 1 no retorna cap resultat.

Vegem un exemple senzill per a entendre'n el funcionament.

Exemple de funcionament de consulta recursiva

Per a aquest exemple utilitzarem les dades de la taula d'empleats i la consulta recursiva presentades en l'exemple anterior. Per a entendre l'avaluació de les consultes recursives, seguim els passos definits anteriorment:

- Avaluem la part no recursiva i emmagatzemem les dades a la taula intermèdia i la taula de treball. Els resultats que contenen ambdues taules es poden veure a continuació:

Taula Intermedia			
Id Empleat	Nom	Id Supervisor	Resultat
10	Victoria Setan		Victoria Setan

Taula de Treball			
Id Empleat	Nom	Id Supervisor	Resultat
10	Victoria Setan		Victoria Setan

- A continuació, comencem amb el procés iteratiu.

Iteració 1

S'avalua la part recursiva substituint la crida a la consulta recursiva per les dades que s'emmagatzemen a la taula de treball. Els resultats d'aquesta substitució i avaluació són els següents.

Part Recursiva			
Id Empleat	Nom	Id Supervisor	Resultat
7	Victoria Suárez	10	Victoria Setan <- Victoria Suárez
8	Víctor Anllada	10	Victoria Setan <- Víctor Anllada
9	José María Llopis	10	Victoria Setan <- José María Llopis

Vegeu que el que s'obté en aquesta part de la iteració són els empleats que tenen com a supervisor els empleats que existeixen a la taula de treball.

A continuació, s'aplica la clàusula `UNION ALL` (tal com s'ha especificat a la consulta) entre aquests resultats i els existents a la taula intermèdia, i es guarden en aquesta última.

Com a últim pas, s'eliminen les dades de la taula de treball i s'afegeixen les obtingudes en aquest pas. En finalitzar aquesta iteració, la taula intermèdia i la de treball apareixen ara amb les següents dades:

Taula Intermèdia			
Id Empleat	Nom	Id Supervisor	Resultat
10	Victoria Setan		Victoria Setan
7	Victoria Suárez	10	Victoria Setan <- Victoria Suárez
8	Víctor Anllada	10	Victoria Setan <- Víctor Anllada
9	José María Llopis	10	Victoria Setan <- José María Llopis

Taula de Treball			
Id Empleat	Nom	Id Supervisor	Resultat
7	Victoria Suárez	10	Victoria Setan <- Victoria Suárez
8	Víctor Anllada	10	Victoria Setan <- Víctor Anllada
9	José María Llopis	10	Victoria Setan <- José María Llopis

Iteració 2

De nou s'avalua la part recursiva, igual que la iteració anterior, que proporciona els següents resultats.

Part Recursiva			
Id Empleat	Nom	Id Supervisor	Resultat
1	Manuel Vázquez	7	Victoria Setan <- Victoria Suárez <- Manuel Vázquez
4	Alejandra Martínez	7	Victoria Setan <- Victoria Suárez <- Alejandra Martínez
11	Manuel Bertrán	9	Victoria Setan <- José María Llopis <- Manuel Bertrán

S'obtenen, per tant, els empleats que tenen com a supervisor Victoria Suárez, Víctor Anllada o José María Llopis, que són els empleats que apareixen ara a la taula de treball.

A continuació, s'aplica de nou `UNION ALL` entre els resultats obtinguts en aquesta iteració i els existents a la taula intermèdia, que es guarden en aquesta última.

Com a últim pas, s'eliminen les dades de la taula de treball i s'afegeixen aquests resultats, cosa que deixa la taula intermèdia i la de treball en el següent estat:

Taula Intermèdia			
Id Empleat	Nom	Id Supervisor	Resultat
10	Victoria Setan		Victoria Setan
7	Victoria Suárez	10	Victoria Setan <- Victoria Suárez
8	Víctor Anllada	10	Victoria Setan <- Víctor Anllada
9	José María Llopis	10	Victoria Setan <- José María Llopis
1	Manuel Vázquez	7	Victoria Setan <- Victoria Suárez <- Manuel Vázquez
4	Alejandra Martínez	7	Victoria Setan <- Victoria Suárez <- Alejandra Martínez
11	Manuel Bertrán	9	Victoria Setan <- José María Llopis <- Manuel Bertrán

Taula de Treball			
Id Empleat	Nom	Id Supervisor	Resultat
1	Manuel Vázquez	7	Victoria Setan <- Victoria Suárez <- Manuel Vázquez
4	Alejandra Martínez	7	Victoria Setan <- Victoria Suárez <- Alejandra Martínez
11	Manuel Bertrán	9	Victoria Setan <- José María Llopis <- Manuel Bertrán

Iteració 3

En la següent iteració, s'aplica exactament el mateix mètode que en les anteriors; ara s'intenta obtenir els empleats que tenen com a supervisor Manuel Vázquez, Alejandra Martínez o Manuel Bertrán, que són els següents:

Part recursiva			
Id Empleat	Nom	Id Supervisor	Resultat
2	Elena Rodríguez	1	Victoria Setan <- Victoria Suárez <- Manuel Vázquez <- Elena Rodríguez
3	José Pérez	1	Victoria Setan <- Victoria Suárez <- Manuel Vázquez <- José Pérez
5	Marina Rodríguez	4	Victoria Setan <- Victoria Suárez <- Alejandra Martínez <- Marina Rodríguez
6	Fernando Nadal	4	Victoria Setan <- Victoria Suárez <- Alejandra Martínez <- Fernando Nadal

Tornem a aplicar UNION ALL i guardem els resultats obtinguts a la taula intermèdia de nou. Finalment, com ja hem fet abans, eliminem les dades de la taula de treball i afegim a aquesta taula els resultats obtinguts en aquesta iteració.

Les taules intermèdia i de treball apareixen ara amb les següents dades:

Taula Intermèdia			
Id Empleat	Nom	Id Supervisor	Resultat
10	Victoria Setan		Victoria Setan
7	Victoria Suárez	10	Victoria Setan <- Victoria Suárez
8	Víctor Anllada	10	Victoria Setan <- Víctor Anllada
9	José María Llopis	10	Victoria Setan <- José María Llopis
1	Manuel Vázquez	7	Victoria Setan <- Victoria Suárez <- Manuel Vázquez
4	Alejandra Martínez	7	Victoria Setan <- Victoria Suárez <- Alejandra Martínez
11	Manuel Bertrán	9	Victoria Setan <- José María Llopis <- Manuel Bertrán
2	Elena Rodríguez	1	Victoria Setan <- Victoria Suárez <- Manuel Vázquez <- Elena Rodríguez
3	José Pérez	1	Victoria Setan <- Victoria Suárez <- Manuel Vázquez <- José Pérez
5	Marina Rodríguez	4	Victoria Setan <- Victoria Suárez <- Alejandra Martínez <- Marina Rodríguez
6	Fernando Nadal	4	Victoria Setan <- Victoria Suárez <- Alejandra Martínez <- Fernando Nadal

Taula de Treball			
Id Empleat	Nom	Id Supervisor	Resultat
2	Elena Rodríguez	1	Victoria Setan <- Victoria Suárez <- Manuel Vázquez <- Elena Rodríguez
3	José Pérez	1	Victoria Setan <- Victoria Suárez <- Manuel Vázquez <- José Pérez
5	Marina Rodríguez	4	Victoria Setan <- Victoria Suárez <- Alejandra Martínez <- Marina Rodríguez
6	Fernando Nadal	4	Victoria Setan <- Victoria Suárez <- Alejandra Martínez <- Fernando Nadal

Iteració 4

S'avalua de nou la part recursiva, i, en aquest cas, no obtenim cap resultat, ja que els empleats que hi ha a la taula de treball no tenen cap empleat al seu càrrec.

Com que no hi ha cap fila que s'hagi de processar, una vegada arribats a aquest punt, la taula de treball es destrueix i els resultats que conté la taula intermèdia seran els resultats que processarà el SELECT de la consulta principal (que en aquest cas és SELECT nom, resultat FROM jerarquies ORDER BY resultat). La consulta, per tant, retornarà el següent conjunt de dades (vegeu que els resultats han estat ordenats de manera diferent de com s'han obtingut):

Nom	Resultat
Victoria Setan	Victoria Setan
José María Llopis	Victoria Setan <- José María Llopis
Manuel Bertrán	Victoria Setan <- José María Llopis <- Manuel Bertrán
Víctor Anllada	Victoria Setan <- Víctor Anllada
Victoria Suárez	Victoria Setan <- Victoria Suárez
Alejandra Martínez	Victoria Setan <- Victoria Suárez <- Alejandra Martínez
Fernando Nadal	Victoria Setan <- Victoria Suárez <- Alejandra Martínez <- Fernando Nadal
Marina Rodríguez	Victoria Setan <- Victoria Suárez <- Alejandra Martínez <- Marina Rodríguez
Manuel Vázquez	Victoria Setan <- Victoria Suárez <- Manuel Vázquez
Elena Rodríguez	Victoria Setan <- Victoria Suárez <- Manuel Vázquez <- Elena Rodríguez
José Pérez	Victoria Setan <- Victoria Suárez <- Manuel Vázquez <- José Pérez

Quan es treballa amb consultes recursives, és molt important assegurar-se que, en algun moment, la part recursiva no retorna cap fila o correm el risc d'entrar en un bucle infinit.

El que ens pot ajudar a provar consultes que puguin acabar en un bucle infinit és limitar els resultats de la consulta utilitzant la clàusula `LIMIT N` (on `N` és el nombre de files que la consulta pot retornar), sempre que les dades de la consulta principal no s'ordenin o es combinin amb dades d'altres taules.

Exemple de bucle infinit en una consulta recursiva

Suposem que, en lloc d'especificar amb un valor nul la falta d'un supervisor, això s'especifica amb l'assignació de l'identificador del mateix empleat. D'aquesta manera, el director general (Victoria Setan) tindria com a identificador de supervisor el seu propi identificador d'empleat. Per tant, necessitem canviar la nostra consulta per la que es mostra a continuació:

```
WITH RECURSIVE jerarquies AS (
    SELECT
        id_empleat,
        nom,
        id_supervisor,
        CAST (nom AS TEXT) AS resultat
    FROM
        empleat
    WHERE
        id_supervisor = 10
    UNION ALL
    SELECT
        e.id_empleat,
        e.nom,
        e.id_supervisor,
        CAST (j.resultat || ' <- ' || e.nom AS TEXT) AS resultat
    FROM
        empleat e INNER JOIN jerarquies j
        ON (e.id_supervisor = j.id_empleat )
)
SELECT
    nom,
    resultat
FROM
    jerarquies
ORDER BY
    resultat
```

Exercici

Intenteu entendre per què es produeix un bucle infinit utilitzant els passos que s'han explicat anteriorment. Intenteu també donar una solució al problema.

Si executem aquesta consulta, veurem que entrarem en un bucle infinit, cosa que provocarà que l'SGBD mai no ens tornni un resultat. Si volem veure si tenim raó, només cal executar la consulta sense la clàusula `ORDER BY resultat`, i afegir, per exemple, la clàusula `LIMIT 1000`, i podrem veure el que la consulta genera a la taula intermèdia.

2.3.2. Consultes CTE amb sentències de manipulació de dades

Com ja hem esmentat anteriorment, PostgreSQL permet definir sentències de manipulació de dades en consultes CTE, tant en la definició de consultes auxiliars amb `WITH` com a la consulta principal. Això ens pot ser útil per a executar diverses operacions en una mateixa consulta.

Les sentències DML que formen part del `WITH` s'executen exactament una única vegada, i sempre fins a ser completades, independentment que la consulta principal faci referència a les dades retornades. Aquesta funcionalitat és diferent de l'especificada anteriorment quan la consulta dins del `WITH` és un `SELECT`, ja que aquest s'executa només si és cridat des de la consulta principal.

Vegem un exemple d'aquesta funcionalitat.

Exemple de funcionament de consulta CTE amb INSERT i DELETE

Imaginem que volem eliminar els empleats amb identificador 2 i 3 (Elena Rodríguez i José Pérez respectivament) perquè ja no treballen a la nostra empresa, i moure'ls a una taula d'històric d'empleats (`empleat_hist`). Podem pensar a fer aquestes dues operacions mitjançant dues sentències SQL: la primera, una inserció d'aquests empleats a la taula d'històric, i la segona, esborrament d'aquests empleats a la taula mestra, ambdues operacions com a part d'una transacció.

```
START TRANSACTION;

INSERT INTO empleat_hist
SELECT * FROM empleat WHERE id_empleat IN (2, 3);

DELETE FROM empleat WHERE id_empleat IN (2, 3);

COMMIT;
```

Transaccions

Parlarem de les transaccions en l'última secció d'aquest mòdul.

Common table expression a PostgreSQL

Per a més informació sobre les CTE a PostgreSQL 9.3, visiteu l'enllaç següent: <http://www.postgresql.org/docs/9.3/static/queries-with.html>

Aquesta mateixa operació podríem haver-la fet amb una consulta CTE:

```
WITH empleats_baixa AS (
    DELETE
    FROM
        empleat
    WHERE
        id_empleat IN (2, 3)
        RETURNING *
)
INSERT INTO empleat_hist
SELECT
    *
FROM
    empleats_baixa
```

La definició de `DELETE` s'encarrega d'eliminar els empleats que han causat baixa i de retornar aquestes files eliminades mitjançant la clàusula `RETURNING *`, que són les files que es retornen en avaluar `empleats_baixa`. Aquestes files retornades són les que s'utilitzen per a la inserció en `empleat_hist`.

Si, en canvi, la sentència DML dins d'`empleats_baixa` no retorna resultats (mitjançant `RETURNING`), llavors aquesta no podria ser referenciada per la consulta principal. Així, la següent consulta (vegeu que hem eliminat `RETURNING *`) ens retornaria el següent error:

```
WITH empleats_baixa AS (
    DELETE
    FROM empleat
    WHERE
        id_empleat IN (2, 3)
)
INSERT INTO empleat_hist
SELECT
    *
FROM empleats_baixa
```

```
ERROR: WITH query "empleats_baixa" does not have a RETURNING clause
LINE 10:     empleats_baixa
                  ^
*****
***** Error *****

ERROR: WITH query "empleats_baixa" does not have a RETURNING clause
SQL state: 0A000
Character: 145
```

3. Funcions analítiques

Els SGBD relacionals han inclòs, des dels seus inicis, funcionalitat per a permetre l'agregació de dades. Aquesta funcionalitat, que s'implementa mitjançant clàusules `GROUP BY` i funcions d'agregació com `MAX`, `SUM` i `AVG` entre altres, ens permet presentar una visió de les dades de manera agregada. També ens permet especificar condicions de cerca una vegada agregades les dades amb la clàusula `HAVING`. Per exemple, ens permetria obtenir la mitjana del salari entre els empleats de l'empresa agrupada pel lloc de treball dels empleats per a obtenir el salari mitjà per lloc. Malgrat aquesta útil funcionalitat, els SGBD encara tenen la limitació de visualitzar les dades de dues maneres: dades en brut (sense processar) o dades agregades (mitjançant les funcions d'agregació, com hem comentat). Com podríem presentar tots dos tipus d'informació conjuntament? Per a això, com a part de l'estàndard SQL:1999, s'han introduït les **funcions analítiques**.

Les funcions analítiques amplien el llenguatge SQL de manera que ens permeten fer ànàlisis més complexos alhora que consultem les dades en brut sense necessitat d'agregar-les mitjançant l'accés a les dades d'altres files que formen part de la consulta. Teòricament, no hi ha res que les funcions analítiques facin que no pugui fer-se mitjançant consultes SQL complexes, subconsultes o operacions de combinació, però sí que ens permeten fer els mateixos càlculs d'una manera molt més senzilla i elegant (menys línies de codi), a més d'utilitzar funcionalitat nativa de l'SGBD que està implementada per a donar un rendiment òptim.

Veurem a continuació amb més detall què són les funcions analítiques, els beneficis que aquestes ens aporten i quin tipus de problemes ens solucionen, i també les funcions analítiques més comunes a PostgreSQL.

3.1. Concepte

Les **funcions analítiques** (també denominades *window functions*) s'utilitzen per a fer càlculs dins d'un context de manera que una fila vegi i utilizi dades més enllà de les pertanyents a aquesta fila.

Definim **finestra** com el context en el qual la funció analítica ha de fer el càlcul específicat. En altres paraules, defineix quines altres files s'han de tenir en consideració (a més de la fila actual).

Les funcions analítiques fan càlculs sobre un conjunt de files que, d'alguna manera, es relacionen amb el que es denomina la fila actual. És a dir, la funció analítica és capaç d'accedir a informació d'altres files des de la fila que es llegeix o processa. Per a aclarir aquesta definició, vegem el següent exemple:

Exemple de funció analítica

La següent taula Alumne mostra les dades d'un conjunt d'alumnes donats d'alta en l'assignatura *Introducció a les bases de dades* impartida a la UOC.

Alumne					
<u>Id_Alumne</u>	<u>Nom / Cognoms</u>	<u>Ciutat</u>	<u>Edat</u>	<u>N. Llibres</u>	<u>N. Assignatures</u>
1	Manuel Vázquez	Barcelona	24	10	7
2	Elena Rodríguez	Barcelona	22	6	3
3	José Pérez	Barcelona	25	4	9
4	Alejandra Martínez	Barcelona	18	11	4
5	Marina Rodríguez	Tarragona	19	9	4
6	Fernando Nadal	Tarragona	21	8	5
7	Victoria Suárez	Tarragona	20	6	8
8	Víctor Anllada	Lleida	23	7	7
9	Felisa Sánchez	Lleida	25	5	2
10	José María Llopis	Lleida	18	5	4
11	Victoria Setan	Lleida	23	5	2
12	Wenceslao Fernández	Lleida	18	6	1

Utilitzant aquestes dades com a base, suposem que volem obtenir un llistat de ciutats, alumnes i nombre d'assignatures ordenat per ciutat i nom d'alumne ascendentment. Per a cadascun dels alumnes, volem mostrar la mitjana d'assignatures de què s'han matriculat tots els alumnes que resideixen a la mateixa ciutat que l'alumne en qüestió. Una proposta de consulta podria ser la següent:

```

SELECT
    a1.ciutat,
    a1.nom_cognoms,
    a1.n_assignatures,
    a3.mitjana
FROM
    alumne a1,
    (
        SELECT
            a2.ciutat,
            AVG(a2.n_assignatures) AS mitjana
        FROM
            alumne a2
        GROUP BY
            a2.ciutat
    ) a3
WHERE
    a1.ciutat = a3.ciutat
ORDER BY
    a1.ciutat ASC,
    a1.nom_cognoms ASC

```

Els resultats d'aquesta consulta es poden veure en la següent taula. Com podeu veure a la consulta proposada, hem de crear una subconsulta sobre la taula Alumne (denominada a3) que calculi primer la mitjana d'assignatures per ciutat, i fer a continuació una operació de combinació (`JOIN`) entre a3 i la taula Alumne (denominada a1).

Ciutat	Nom / Cognoms	N. Assignatures	Mitjana
Barcelona	Alejandra Martínez	4	5.75
Barcelona	Elena Rodríguez	3	5.75
Barcelona	José Pérez	9	5.75
Barcelona	Manuel Vázquez	7	5.75
Lleida	Felisa Sánchez	2	3.2
Lleida	José María Llopis	4	3.2
Lleida	Víctor Anllada	7	3.2
Lleida	Victoria Setan	2	3.2
Lleida	Wenceslao Fernández	1	3.2
Tarragona	Fernando Nadal	5	5.666666667
Tarragona	Marina Rodríguez	4	5.666666667
Tarragona	Victoria Suárez	8	5.666666667

Aquesta manera de crear consultes, si bé ens retorna els resultats esperats, requereix més processament de dades per part de l'SGBD, ja que ha de calcular primer la mitjana per ciutat i després combinar-la amb la mateixa taula Alumne. A més, codificar, interpretar i mantenir consultes d'aquest estil pot arribar a ser bastant enutjós.

Vegem el que les funcions analítiques ens permeten fer. La següent consulta fa la mateixa operació que la consulta anterior sense necessitat de crear subconsultes.

```

SELECT
    ciutat,
    nom_cognoms,
    n_assignatures,
    AVG(n_assignatures) OVER (PARTITION BY ciutat) AS mitjana
FROM
    alumne
ORDER BY
    ciutat ASC,
    nom_cognoms ASC

```

Els resultats, com s'esperava, són els mateixos que en la consulta anterior, amb la diferència que el codi és molt més lleigible i no requereix tant d'esforç a l'hora de codificar la consulta, a més de ser més eficient a l'hora de processar les dades.

Després de l'exemple vist, podem aprofundir una mica més en aquest tipus de funcions. Com s'ha vist en l'exemple, la crida a funcions analítiques es fa utilitzant un format especial, que genèricament, es podria definir com segueix:

```

function_name (arg) OVER (
    [PARTITION BY ...]
    [ORDER BY ...]
    [ [ROWS | RANGE] ... ]
)

```

La clàusula `OVER` és el que ens permet distingir si s'estan aplicant funcions analítiques o funcions d'agregació. Aquesta clàusula ens permet definir com dividir i ordenar el conjunt de dades per al seu posterior processament per la funció.

Aquesta divisió de les dades es fa mitjançant la clàusula `PARTITION BY`, que crea el que anteriorment definim com a **fines tra**, és a dir, una sèrie de particions que seran tractades separadament per la funció analítica. Per a cada fila dins de cada partició s'aplica la funció analítica, tenint en consideració la resta de files que pertanyen a aquesta partició per a fer el càlcul desitjat.

També es pot controlar l'ordre de processament de les files en cada partició mitjançant la clàusula `ORDER BY`. Aquesta ordenació de les dades **dins de la partició** no afecta (i no ha de ser la mateixa que) l'**ordenació en la qual les dades finals es mostren** després de l'execució de la consulta.

Les clàusules esmentades anteriorment, segons si és o no necessari, es poden ometre de la crida:

- En el cas que la clàusula `PARTITION BY` no es defineixi, es consideraran totes les files del resultat de la consulta com una sola partició.
- Quan s'ometi la clàusula `ORDER BY`, les files es processaran sense un ordre definit, la qual cosa podria afectar el resultat final, dependent de la funció analítica que s'utilitzi.

Un altre concepte rellevant que convé considerar quan treballem amb funcions analítiques és el de **marc**, que definim a continuació.

El **marc** (*frame*) es defineix com un subconjunt de files dins d'una partició. Moltes de les funcions analítiques (no totes) actuen dins del marc definit en lloc d'actuar sobre el conjunt de dades de la partició.

En altres paraules, el marc permet definir una **subfinestra dins de la finestra** establerta mitjançant la clàusula `PARTITION BY`. El marc permet canviar el context de la finestra de manera **dinàmica** (el marc incrementa/reduïx les files que cal tenir en compte en relació amb la fila que es processa) o bé estableix un context de finestra **estàtic** (un límit inferior i un límit superior fixos). El marc es pot definir mitjançant les clàusules `RANGE` o `ROWS`.

Diferència entre finestra i marc

Suposem que tenim la següent consulta que ens proporciona la **suma acumulativa** de les assignatures dels alumnes que viuen a cada ciutat. La consulta SQL necessària seria la següent:

```
SELECT
    ciutat,
    nom_cognoms,
    n_assignatures,
    SUM(n_assignatures) OVER
        (PARTITION BY ciutat ORDER BY nom_cognoms ASC
         RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS suma
FROM
    alumne
ORDER BY
    ciutat ASC,
    nom_cognoms ASC
```

Els resultats obtinguts es poden veure a la taula inferior.

Ciutat	Nom / Cognoms	N. Assignatures	Suma
Barcelona	Alejandra Martinez	4	4
Barcelona	Elena Rodriguez	3	7
Barcelona	José Pérez	9	16
Barcelona	Manuel Vázquez	7	23
Lleida	Felisa Sánchez	2	2
Lleida	José María Llopis	4	6
Lleida	Victor Anllada	7	13
Lleida	Victoria Setan	2	15
Lleida	Wenceslao Fernández	1	16
Tarragona	Fernando Nadal	5	5
Tarragona	Marina Rodríguez	4	9
Tarragona	Victoria Suarez	8	17

Suma acumulativa

Definim com a **suma acumulativa** (*running sum, cumulative sum o running total* en anglès) com la suma en seqüència de nombres, el valor dels quals s'actualitza cada vegada que un nombre s'afegeix a la seqüència, sumant aquest nombre a la suma acumulativa prèvia.

En aquest cas, la **finestra** de la consulta es defineix a partir de `CIUTAT`, mitjançant la clàusula `PARTITION BY CIUTAT`, de manera que s'obtenen tres particions de dades: els que viuen a Barcelona, els que viuen a Lleida i els que viuen a Tarragona.

Cadascuna d'aquestes particions s'ordena alfabèticament i ascendentment per nom d'alumne (clàusula ORDER BY NOM_COGNOMS ASC com a part d'OVER). No s'ha de confondre amb com s'ordenen les dades de sortida (ORDER BY CIUTAT ASC, NOM_COGNOMS ASC).

Finalment, el marc s'ha definit com a RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW. Això significa que el marc es defineix des de l'inici de la partició fins a la fila actual. Utilitzant la partició de Barcelona com a exemple, això significa que:

- Per a l'alumna Alejandra Martínez el marc inclou només aquesta fila (és la primera fila de la partició, i com a límit superior és la fila actual), per la qual cosa la suma acumulativa inclou només el nombre d'assignatures d'aquesta alumna.

Marc de l'alumne «Alejandra Martínez»			
Ciutat	Nom / Cognoms	N. Assignatures	Suma
Barcelona	Alejandra Martínez	4	4

- Per a l'alumna Elena Rodríguez el marc inclou des de l'inici de la partició (Alejandra Martínez) fins a la fila actual que es processa (Elena Rodríguez), per la qual cosa la suma acumulativa inclou el nombre d'assignatures d'aquesta alumna més les d'Alejandra Martínez, que sumen un total de 7 assignatures ($4 + 3$).

Marc de l'alumne «Elena Rodríguez»			
Ciutat	Nom / Cognoms	N. Assignatures	Suma
Barcelona	Alejandra Martínez	4	4
Barcelona	Elena Rodríguez	3	7

- Per a l'alumne José Pérez el marc inclou des de l'inici de la partició (Alejandra Martínez) fins a la fila actual que es processa (José Pérez), per la qual cosa la suma acumulativa inclou el nombre d'assignatures d'aquest alumne més les d'Elena Rodríguez i Alejandra Martínez, que sumen un total de 16 assignatures ($4 + 3 + 9$).

Marc de l'alumne «José Pérez»			
Ciutat	Nom / Cognoms	N. Assignatures	Suma
Barcelona	Alejandra Martínez	4	4
Barcelona	Elena Rodríguez	3	7
Barcelona	José Pérez	9	16

- Finalment, per a l'alumne Manuel Vázquez el marc inclou des de l'inici de la partició (Alejandra Martínez) fins a la fila actual que es processa (Manuel Vázquez), per la qual cosa la suma acumulativa inclou el nombre d'assignatures d'aquest alumne més les de José Pérez, Elena Rodríguez i Alejandra Martínez, que sumen un total de 16 assignatures ($4 + 3 + 9 + 7$).

Marc de l'alumne «Manuel Vázquez»			
Ciutat	Nom / Cognoms	N. Assignatures	Suma
Barcelona	Alejandra Martínez	4	4
Barcelona	Elena Rodríguez	3	7
Barcelona	José Pérez	9	16
Barcelona	Manuel Vázquez	7	23

Finalment, és important destacar que les funcions analítiques només poden executar-se com a part de les clàusules `SELECT` i `ORDER BY` dins d'una consulta, és a dir, no se'n permet la utilització en clàusules `WHERE`, `GROUP BY` o `HAVING`, ja que les funcions analítiques es processen després que aquestes clàusules s'hagin completat.

3.2. Beneficis de les funcions analítiques

Entre els possibles beneficis de l'ús de funcions analítiques destaquen els següents:

- 1) Facilitar l'obtenció de càlculs complexos en informes i processos ETL de manera més senzilla, complexitat que sol donar-se molt sovint dins de l'àmbit dels magatzems de dades (*data warehouse*).
- 2) Millorar el rendiment de les consultes SQL: consultes que abans requerien operacions de combinació sobre la mateixa taula es poden implementar amb clàusules SQL molt més senzilles.
- 3) Proporcionar una manera més clara i concisa de generar consultes SQL, la qual cosa facilita el manteniment del codi i incrementa la productivitat dels desenvolupadors.
- 4) La sintaxi de funcions analítiques forma part de l'SQL estàndard, la qual cosa significa que estan suportades per multitud d'SGBD del mercat, entre els quals tenim PostgreSQL i Oracle.

Sobre la base del que s'ha comentat, i a mode de resum, podem afirmar que les funcions analítiques ens faciliten el càlcul, de manera eficient i elegant, sobre un conjunt de files per a retornar-nos un valor relacionat amb un subconjunt de dades d'aquesta consulta.

A continuació, veurem com es fan les crides de funcions analítiques a PostgreSQL, i també la sintaxi necessària i les seves diferents regles.

3.3. Funcions analítiques a PostgreSQL

La crida a funcions analítiques a PostgreSQL es pot fer utilitzant qualsevol de les següents formes:

```
function_name ([expression [, expression ... ]]) OVER (window_definition)
function_name (* ) OVER (window_definition)
```

En aquestes definicions, `expression` representa qualsevol expressió que no contingui una crida a una funció analítica: podria tractar-se d'una columna d'una taula, una funció d'agregació, una constant o un càlcul, entre altres.

La clàusula `window_definition` ens permet definir la finestra mitjançant la següent sintaxi:

```
[ existing_window_name ]
[ PARTITION BY expression [, ...] ]
[ ORDER BY expression [ ASC | DESC | USING operator ]
          [ NULLS { FIRST | LAST } ] [, ...] ]
[ frame_clause ]
```

Com hem vist anteriorment, les clàusules `PARTITION BY` i `ORDER BY` ens serveixen per a definir les particions i com les dades dins de cada partió seran ordenades. Per la seva banda, la clàusula opcional `frame_clause` permet que la funció analítica treballi dins d'un **marc**. Aquesta clàusula es pot especificar mitjançant qualsevol de les dues següents opcions:

```
{ RANGE | ROWS } frame_start
{ RANGE | ROWS } BETWEEN frame_start AND frame_end
```

on els possibles valors per als paràmetres `frame_start` i `frame_end`, que delimiten l'àmbit del marc, poden ser alguna de les següents opcions:

```
UNBOUNDED PRECEDING
offset PRECEDING
CURRENT ROW
offset FOLLOWING
UNBOUNDED FOLLOWING
```

Com s'ha comentat anteriorment, el marc pot especificar-se mitjançant rang (RANGE) o mitjançant files (ROWS). RANGE permet definir rangs de files utilitzant els delimitadors UNBOUNDED PRECEDING, CURRENT ROW i UNBOUNDED FOLLOWING, sense poder accedir a una posició específica dins de la parti-

Offset

Distància o posició relativa a un element.

ció. D'això últim se n'encarrega `ROWS`, és a dir, permet no solament especificar rangs mitjançant els delimitadors que permet `RANGE`, sinó que a més permet definir marcs utilitzant posicions concretes dins de la partició mitjançant la definició d'un *offset*.

Si el paràmetre `frame_start` es defineix com a `UNBOUNDED PRECEDING` llavors significa que el marc comença amb la primera fila de la partició, i de manera similar, si `frame_end` es defineix com a `UNBOUNDED FOLLOWING`, llavors significa que el marc acaba a l'última fila de la partició. Si `frame_end` no s'especifica, llavors el valor per defecte és la fila que s'està processant (`CURRENT ROW`). Per defecte, en el cas que s'ometi la definició del marc, l'opció seleccionada per PostgreSQL és `RANGE UNBOUNDED PRECEDING`, que és equivalent a especificar `RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW`.

Si s'especifica `ORDER BY` en la definició de la finestra, llavors el marc s'estableix des de l'inici de la partició fins a l'última fila representativa del conjunt de files equivalent definit per `ORDER BY`. Si no s'especifica `ORDER BY`, totes les files de la partició s'inclouen dins del marc, ja que totes es consideren representatives o iguals a la fila processada.

En mode `RANGE`, si el paràmetre `frame_start` es defineix com a `CURRENT ROW`, això significa que el marc comença amb la primera fila representativa del conjunt de files equivalent que defineix la clàusula `ORDER BY`, mentre que si el paràmetre `frame_end` es defineix com a `CURRENT ROW`, això significa que el marc acaba a l'última fila representativa del conjunt de files equivalent que defineix la clàusula `ORDER BY`.

Exemple de RANGE

Com a exemple d'ús de `RANGE`, tenim la següent consulta utilitzada anteriorment, que ens proporciona la suma acumulativa de les assignatures dels alumnes que viuen a cada ciutat. Aquesta consulta utilitza la clàusula `RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW`, la qual cosa ens indica que el marc de la fila s'estableix des de l'inici de la partició fins a la fila actual.

```
SELECT
    ciutat,
    nom_cognoms,
    n_assignatures,
    SUM(n_assignatures) OVER
        (PARTITION BY ciutat ORDER BY nom_cognoms ASC
         RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS suma
FROM
    alumne
ORDER BY
    ciutat ASC,
    nom_cognoms ASC
```

En mode `ROWS`, quan s'especifica `CURRENT ROW` a la definició del marc, significa que el marc comença o acaba amb la fila actual. En canvi, en mode `RANGE` significa que el marc comença o acaba amb la primera o última fila del conjunt de files representatives depenen de com estan ordenades mitjançant `ORDER BY`.

És important destacar que l'ús de la clàusula `ROWS` podria produir resultats inesperats si la clàusula `ORDER BY` no ordena les files de manera única. Les opcions de `RANGE` estan precisament dissenyades per a assegurar que les files aparellades sobre la base de l'`ORDER BY` siguin tractades de manera similar: qualsevol parell de files que estiguin aparellades pertanyen o no al marc.

Les clàusules `offset PRECEDING` i `offset FOLLOWING` només estan disponibles en mode `ROWS`. Això indica que el marc comença amb la fila que es troba en la posició `offset` abans de la fila actual (`offset PRECEDING`) i acaba amb la fila que es troba en la posició `offset` després de la fila actual (`offset FOLLOWING`). El valor d'`offset` ha de ser una expressió entera que no contingui variables, ni funcions d'agregació ni analítiques. El valor tampoc no pot ser nul o negatiu, si bé pot ser zero, cosa que indica que es tracta de la fila actual (`CURRENT ROW`).

Exemple d'ús de clàusula `ROWS` amb `offset`

Utilitzarem com a base la consulta de suma acumulativa d'assignatures que hem vist fins ara. La nova consulta definida utilitza la clàusula `ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING`, la qual cosa ens indica que el marc de la fila s'estableix des d'una fila anterior a la fila actual fins a una fila posterior a la fila actual (en aquest cas, tots dos `offset` s'han especificat com a 1).

```
SELECT
    ciutat,
    nom_cognoms,
    n_assignatures,
    SUM(n_assignatures) OVER (PARTITION BY ciutat
                                ORDER BY nom_cognoms ASC
                                ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS suma
FROM
    alumne
ORDER BY
    ciutat ASC,
    nom_cognoms ASC
```

Els resultats d'aquesta consulta seran els següents. La suma que apareix a cada fila s'obté amb la suma de les assignatures de l'alumne immediatament anterior a la fila actual, de l'alumne immediatament posterior a la fila actual, i les assignatures de la fila actual, sempre dins de la partició establet (CIUTAT).

Ciutat	Nom / Cognoms	N. Assignatures	Suma
Barcelona	Alejandra Martínez	4	7
Barcelona	Elena Rodríguez	3	16
Barcelona	José Pérez	9	19
Barcelona	Manuel Vázquez	7	16
Lleida	Felisa Sánchez	2	6
Lleida	José María Llopis	4	13
Lleida	Víctor Anllada	7	13
Lleida	Victoria Setan	2	10
Lleida	Wenceslao Fernández	1	3
Tarragona	Fernando Nadal	5	9
Tarragona	Marina Rodríguez	4	17
Tarragona	Victoria Suárez	8	12

Prenent com a exemple l'alumna Elena Rodríguez: la suma de les assignatures és 16, perquè és la suma de les assignatures d'Alejandra Martínez (fila anterior, 4 assignatures), José Pérez (fila posterior, 3 assignatures) i Elena Rodríguez (fila actual, 3 assignatures).

Hi ha una sèrie de restriccions sobre la definició de marcs: `frame_start` no pot prendre un valor `UNBOUNDED FOLLOWING` (és a dir, no pot començar amb l'última fila de la partició), `frame_end` no pot prendre un valor `UNBOUNDED PRECEDING` (és a dir, no pot finalitzar amb la primera fila de la partició), i l'opció seleccionada com `frame_end` no pot referir-se a una fila que aparegui abans que el valor de `frame_start` (com a exemple, l'opció `RANGE BETWEEN CURRENT ROW AND value PRECEDING` no estaria permesa).

En el cas que s'usi el format de finestra `existing_window_name`, aquest ha de referir-se a una entrada en la llista de finestres especificada mitjançant la clàusula `WINDOW`. Les crides a funcions analítiques amb finestres definides mitjançant `WINDOW` es fan de la següent manera:

```
function_name ([expression [, expression ... ]]) OVER window_name
function_name ( * ) OVER window_name
```

En aquest cas, `window_name` referencia una finestra especificada mitjançant la clàusula `WINDOW` de PostgreSQL dins de la consulta. Un dels beneficis d'emprar aquesta clàusula és que permet referenciar la mateixa finestra en diverses parts de la consulta, de manera que evitem la duplicitat de clàusules `OVER` i així evitem errors en la definició.

El funcionament d'aquest tipus de crides és el següent: es copia la definició de la finestra definida mitjançant aquesta clàusula, per la qual cosa la finestra nova no pot incloure el seu propi `PARTITION BY`, però sí que pot especificar el

seu propi ORDER BY si la finestra que s'utilitza com a plantilla no el té definit. La nova finestra definida sempre utilitza el seu propi frame_clause, per la qual cosa la finestra existent no pot especificar aquesta clàusula.

La clàusula WINDOW permet no solament crear codis més llegibles, sinó que a més permet reutilitzar les finestres dins d'una mateixa consulta.

La forma que proposa PostgreSQL per a definir finestres mitjançant la clàusula WINDOW és la següent:

```
WINDOW window_name AS ( window_definition ) [, ...]
```

on window_definition utilitza el mateix format de definició de finestres explicat anteriorment.

És important destacar la diferència en l'ús d'OVER en les dues formes que hem vist. En la primera forma, s'utilitza OVER (window_definition), mentre que en el segon s'utilitza OVER window_name (vegeu que el primer utilitza parèntesis i l'altre no). El primer requereix obligatòriament la definició d'una finestra, mentre que en el segon es requereix que aquesta finestra estigui definida com a part de la clàusula WINDOW.

Nota

Per a obtenir més detalls sobre la clàusula WINDOW de PostgreSQL, visiteu el següent vincle:

<http://www.postgresql.org/docs/9.3/static/sql-select.html>

Exemple d'utilització de la clàusula WINDOW

Al principi d'aquesta secció s'ha proposat un exemple per a obtenir un llistat de ciutats, alumnes i nombre d'assignatures ordenat per ciutat i nom d'alumne ascendentment. Per a cadascun dels alumnes, es volia mostrar la mitjana d'assignatures de què s'han matriculat tots els alumnes que resideixen a la mateixa ciutat que l'alumne en qüestió. Una proposta de consulta, utilitzant funcions analítiques, seria la següent:

```
SELECT
    ciutat,
    nom_cognoms,
    n_assignatures,
    AVG(n_assignatures) OVER (PARTITION BY ciutat) AS mitjana
FROM
    alumne
ORDER BY
    ciutat ASC,
    nom_cognoms ASC
```

Aquesta mateixa consulta podria definir-se de la següent manera utilitzant la clàusula WINDOW:

```
SELECT
    ciutat,
    nom_cognoms,
    n_assignatures,
    AVG(n_assignatures) OVER w AS mitjana
FROM
    alumne
WINDOW w AS (PARTITION BY ciutat)
ORDER BY
    ciutat ASC,
    nom_cognoms ASC
```

Finalment, i per a acabar aquest apartat sobre funcions analítiques a PostgreSQL, és important esmentar que aquest SGBD permet definir les seves pròpies funcions analítiques utilitzant alguna de les API que proporciona. Encara que és un tema que pot ser interessant per a l'estudiant, s'ha considerat que està fora de l'àmbit d'aquest mòdul didàctic.

3.4. Tipus de funcions analítiques a PostgreSQL

En les següents subseccions veurem alguns exemples de les funcions analítiques més importants que implementa PostgreSQL. Per als exemples, utilitzarem la taula Alumne amb les següents dades:

Alumne					
Id_Alumne	Nom / Cognoms	Ciutat	Edat	N. Llibres	N. Assignatures
1	Manuel Vázquez	Barcelona	24	10	7
2	Elena Rodríguez	Barcelona	22	6	3
3	José Pérez	Barcelona	25	4	9
4	Alejandra Martínez	Barcelona	18	11	4
5	Marina Rodríguez	Tarragona	19	9	4
6	Fernando Nadal	Tarragona	21	8	5
7	Victoria Suárez	Tarragona	20	6	8
8	Víctor Anllada	Lleida	23	7	7
9	Felisa Sánchez	Lleida	25	5	2
10	José María Llopis	Lleida	18	5	4
11	Victoria Setan	Lleida	23	5	2
12	Wenceslao Fernández	Lleida	18	6	1

3.4.1. Row number

Aquesta funció assigna un nombre únic a cada fila dins d'una partició, començant des del valor 1, i seqüencialment segons l'especificació de la clàusula ORDER BY. Aquesta funció es defineix com a `row_number()` i no accepta paràmetres.

Exemple d'utilització de `row_number()`

Utilitzant la taula d'alumnes indicada, obtenir un llistat amb el nom de la ciutat, el nom i cognoms de l'alumne, i la posició de l'alumne a cada ciutat utilitzant el nom i cognoms (alfabèticament) com a criteri de càlcul de posició.

Nota

Consulteu el següent vincle per a obtenir una descripció de totes les funcions analítiques que ofereix PostgreSQL 9.3:
<http://www.postgresql.org/docs/9.3/static/functions-window.html>

```

SELECT
    ciutat,
    nom_cognoms,
    ROW_NUMBER() OVER (PARTITION BY ciutat
                        ORDER BY nom_cognoms ASC) AS rn
FROM
    alumne
ORDER BY
    ciutat ASC,
    nom_cognoms ASC

```

Els resultats d'aquesta consulta seran els següents. Vegeu que per a cada ciutat (PARTITION BY), s'assigna a cadascun dels alumnes un nombre seqüencial –la posició, que s'obté mitjançant row_number() – sobre la base del nom i els cognoms ordenats alfabèticament (ORDER BY).

Ciutat	Nom / Cognoms	RN
Barcelona	Alejandra Martinez	1
Barcelona	Elena Rodriguez	2
Barcelona	José Pérez	3
Barcelona	Manuel Vázquez	4
Lleida	Felisa Sánchez	1
Lleida	José María Llopis	2
Lleida	Víctor Anllada	3
Lleida	Victoria Setan	4
Lleida	Wenceslao Fernández	5
Tarragona	Fernando Nadal	1
Tarragona	Marina Rodriguez	2
Tarragona	Victoria Suarez	3

Aquest tipus de consultes ens pot servir, per exemple, per a obtenir el llistat dels alumnes que es troben en una posició concreta. Per exemple, vegem la consulta necessària per a obtenir els alumnes en la posició número 2 (utilitzant el mateix criteri especificat anteriorment).

```

SELECT
    ds.ciutat,
    ds.nom_cognoms
FROM
(
    SELECT
        ciutat,
        nom_cognoms,
        ROW_NUMBER() OVER (PARTITION BY ciutat
                            ORDER BY nom_cognoms ASC) AS rn
    FROM
        alumne
)
ds
WHERE
    ds.rn = 2
ORDER BY
    ds.ciutat ASC,
    ds.nom_cognoms ASC

```

El resultat d'aquesta consulta és el següent:

Ciutat	Nom / Cognoms
Barcelona	Elena Rodríguez
Lleida	José María Llopis
Tarragona	Marina Rodriguez

Vegeu que, per a poder obtenir els alumnes en la posició 2, hem hagut d'utilitzar subconsultes. La raó principal és que, com s'ha comentat anteriorment, no es permeten funcions analítiques com a part de la clàusula WHERE, per la qual cosa l'ús de subconsultes és necessari per a aplicar condicions sobre els resultats que ens proporcionen aquestes funcions.

3.4.2. Rank

Aquesta funció fa un rànquing de les files d'una partició **amb buits**. En altres paraules, aquesta funció permet classificar els elements d'un grup en posicions (primer, segon, tercer, etc.), i si hi ha elements amb el mateix valor els aparella dins de la mateixa posició, però a l'immediat inferior li dona la posició sobre la base del nombre d'elements existents. Aquesta funció es defineix com a rank() i no accepta paràmetres.

Exemple d'utilització de rank()

Per a aquest exemple, volem obtenir un llistat amb el nom de la ciutat, el nom i cognoms de l'alumne, i el rànquing en nombre d'assignatures de cadascun dels alumnes a cada ciutat. Aquest rànquing s'ha d'obtenir de manera que els alumnes amb més assignatures apareguin amb una posició en el rànquing superior.

```

SELECT
    ciutat,
    nom_cognoms,
    n_assignatures,
    RANK () OVER (PARTITION BY ciutat
                  ORDER BY n_assignatures DESC) AS rk
FROM
    alumne
ORDER BY
    ciutat ASC,
    rk ASC

```

Els resultats que obtenim es mostren en la següent taula: podem veure, utilitzant Lleida com a exemple, que l'alumne amb més assignatures (7) apareix com a primer en el rànquing només per a aquesta ciutat (existeix una partició per a CIUTAT). Vegeu també

la posició dels alumnes Felisa Sánchez i Victoria Setan. Totes dues alumnes tenen 2 assignatures i un rànquing assignat de 3, és a dir, hi ha un empata. El següent en nombre d'assignatures és Wenceslao Fernández, amb una assignatura i un lloc 5 en el rànquing.

Ciutat	Nom / Cognoms	N. Assignatures	RK
Barcelona	José Pérez	9	1
Barcelona	Manuel Vázquez	7	2
Barcelona	Alejandra Martínez	4	3
Barcelona	Elena Rodríguez	3	4
Lleida	Víctor Anllada	7	1
Lleida	José María Llopis	4	2
Lleida	Felisa Sánchez	2	3
Lleida	Victoria Setan	2	3
Lleida	Wenceslao Fernández	1	5
Tarragona	Victoria Suárez	8	1
Tarragona	Fernando Nadal	5	2
Tarragona	Marina Rodríguez	4	3

Pot sorprendre que el rànquing salti de 3 a 5 (com algú pot pensar). Aquesta és la característica de la funció `rank()`: en el cas que hi hagi un empata, el rànquing de la següent fila tindrà assignat el nombre de posició de la fila dins d'aquesta partició, i deixarà un buit entre l'últim rànquing utilitzat en l'empata i el nombre de posició de la fila. En aquest cas, l'alumne Wenceslao Fernández és la fila número 5 dins de la partició de Lleida, i com que és un nou valor dins del rànquing, se li associa el rànquing número 5, i deixa un buit (que seria la posició de rànquing 4).

3.4.3. Dense rank

Aquesta funció fa un rànquing de les files d'una partició **sense buits**. Igual que la funció `rank()`, aquesta funció permet classificar els elements d'un grup en posicions (primer, segon, tercer, etc.). En el cas que hi hagi elements amb el mateix valor, els col·loca dins de la mateixa posició (aparellades) i a l'immediat inferior li dona el correlatiu següent a la classificació de posició. Aquesta funció es defineix com a `dense_rank()` i no accepta paràmetres.

Exemple d'utilització de `dense_rank()`

En aquest cas utilitzarem el mateix requisit que en l'exemple de la funció `rank()`, excepte que emprarem la funció `dense_rank()`.

```

SELECT
    ciutat,
    nom_cognoms,
    n_assignatures,
    DENSE_RANK() OVER (PARTITION BY ciutat
                        ORDER BY n_assignatures DESC) AS rk
FROM
    alumne
ORDER BY
    ciutat ASC,
    rk ASC

```

Els resultats que obtenim són els següents: es pot veure que el llistat obtingut és pràcticament el mateix que en l'exemple proposat anteriorment excepte el rànquing associat a l'alumne Wenceslao Fernández. En aquest cas, aquest alumne té associat un rànquing 4 en lloc de 5, com succeïa amb la funció `rank()`. La raó és que, a diferència de `rank()`, `dense_rank()` no deixa buits en les posicions de rànquing.

Ciutat	Nom / Cognoms	N. Assignatures	RQ
Barcelona	José Pérez	9	1
Barcelona	Manuel Vázquez	7	2
Barcelona	Alejandra Martínez	4	3
Barcelona	Elena Rodríguez	3	4
Lleida	Víctor Anllada	7	1
Lleida	José María Llopis	4	2
Lleida	Victoria Setan	2	3
Lleida	Felisa Sánchez	2	3
Lleida	Wenceslao Fernández	1	4
Tarragona	Victoria Suárez	8	1
Tarragona	Fernando Nadal	5	2
Tarragona	Marina Rodríguez	4	3

3.4.4. Lag

Aquesta funció permet accedir a la informació emmagatzemada en alguna de les files **prèvies** a la fila actual (`CURRENT ROW`) dins de la partició. Aquesta funció es defineix com a `lag()` i accepta els següents paràmetres:

```
lag ( expression [, offset] [, default] );
```

- `expression`: qualsevol valor a avaluar excepte funcions analítiques (per exemple, una columna d'una taula, una funció escalar, etc.).
- `offset` (opcional): indica la posició de la fila prèvia a la fila a què s'accendirà des de la fila actual en la partició. Per exemple, un valor de 3 indica que

s'accedirà a la tercera fila prèvia a la fila actual. Si s'omet, per defecte s'assigna un valor 1 (la fila anterior).

- `default` (opcional): el valor per defecte a assignar en el cas que la fila a què s'ha d'accedir estigui fora dels límits permesos. Si s'omet, per defecte s'assigna un valor `NULL`.

Exemple d'utilització de `lag()`

Per al següent exemple, es demana obtenir un llistat d'alumnes (nom i cognoms), amb la seva edat, i l'edat de l'alumne que es troba dues posicions per darrere, suposant que els alumnes estan ordenats per ordre alfàbetic ascendent. En el cas que no existeixi informació sobre alumnes previs, s'indicarà amb un valor per defecte de -1.

```
SELECT
    nom_cognoms,
    edat,
    LAG(edat, 2, -1)
        OVER (ORDER BY nom_cognoms ASC) AS edat_anterior
FROM
    alumne
ORDER BY
    nom_cognoms ASC,
    edat_anterior ASC
```

Els resultats que obtenim són els següents: els dos primers alumnes (Alejandra Martínez i Elena Rodríguez) no tenen alumnes en dues posicions prèvies alfàbeticament, per la qual cosa s'assigna el valor per defecte especificat en la funció (-1). En canvi, l'alumna Felisa Sánchez té una `EDAT_ANTERIOR = 18`, que és l'edat de l'alumne dues posicions enrere en ordre alfàbetic (això és, Alejandra Martínez).

Nom / Cognoms	Edat	Edat Anterior
Alejandra Martínez	18	-1
Elena Rodríguez	22	-1
Felisa Sánchez	25	18
Fernando Nadal	21	22
José María Llopis	18	25
José Pérez	25	21
Manuel Vázquez	24	18
Marina Rodríguez	19	25
Víctor Anllada	23	24
Victoria Setan	23	19
Victoria Suárez	20	23
Wenceslao Fernández	18	23

Podem observar en aquest exemple que, a diferència dels altres exposats fins ara, no s'ha utilitzat la clàusula `PARTITION BY`. Això és així perquè sobre la base del requisit especificat, tot el conjunt de dades de la taula d'alumnes és considerat com una única partició.

3.4.5. Lead

Al contrari que la funció `lag()`, aquesta funció permet accedir a la informació emmagatzemada en alguna de les files **posteriors** a la fila actual dins de la partició. Aquesta funció es defineix com a `lead()` i accepta els següents paràmetres:

```
lead ( expression [, offset] [, default] );
```

- `expression`: qualsevol valor a avaluar excepte funcions analítiques (per exemple, una columna d'una taula, una funció escalar, etc.).
- `offset` (opcional): indica la posició de la fila posterior a la fila a què s'accedirà des de la fila actual en la partició. Per exemple, un valor de 3 indica que s'acccedirà a la tercera fila posterior a la fila actual. Si s'omet, per defecte s'assigna un valor 1 (la fila següent).
- `default` (opcional): el valor per defecte a assignar en el cas que la fila a què s'ha d'accendir estigui fora dels límits permesos. Si s'omet, per defecte s'assigna un valor `NULL`.

Exemple d'utilització de `lead()`

En aquest exemple, utilitzarem el mateix criteri que en l'exemple utilitzat per a la funció `lag()`, però en lloc de ser dues posicions prèvies, seran dues posicions posteriors i amb el mateix valor per defecte.

```
SELECT
    nom_cognoms,
    edat,
    LEAD(edat, 2, -1) OVER
        (ORDER BY nom_cognoms ASC) AS edat_posterior
FROM
    alumne
ORDER BY
    nom_cognoms ASC,
    edat_posterior ASC
```

Els resultats que obtenim són els següents: per als dos primers alumnes (Alejandra Martínez i Elena Rodríguez), al contrari del que succeïa amb `lag()`, sí que tenim posicions posteriors. Això no és així amb els dos últims alumnes (Victoria Suárez i Wenceslao Fernández), per la qual cosa tenen un valor per defecte de -1.

Nom / Cognoms	Edat	Edat Posterior
Alejandra Martínez	18	25
Elena Rodríguez	22	21
Felisa Sánchez	25	18
Fernando Nadal	21	25
José María Llopis	18	24
José Pérez	25	19
Manuel Vázquez	24	23
Marina Rodríguez	19	23
Víctor Anllada	23	20
Victoria Setan	23	18
Victoria Suárez	20	-1
Wenceslao Fernández	18	-1

Ambdues funcions, `lag()` i `lead()`, ens serveixen per a accedir a dades en diferents posicions dins de la partició especificada sense necessitat de fer operacions de combinació (`JOIN`) amb la mateixa taula.

3.4.6. *First value*

Aquesta funció retorna el valor d'una expressió associat a la primera fila del marc definit a la consulta. Aquesta funció es defineix com a `first_value()` i conté el següent paràmetre:

```
first_value ( expression );
```

- `expression`: qualsevol valor a avaluar excepte funcions analítiques (per exemple, una columna d'una taula, una funció escalar, etc.).

Exemple d'utilització de `first_value()`

Suposem que hem de calcular per a cada alumne la quantitat de llibres màxima entre tots els alumnes de cada ciutat, i també la diferència entre aquest valor i el valor de cadascun dels alumnes. Aquesta consulta podríem crear-la de la següent manera:

```

SELECT
    ciutat,
    nom_cognoms,
    n_llibreria,
    FIRST_VALUE(n_llibreria) OVER
        (PARTITION BY ciutat ORDER BY n_llibreria DESC
         RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS prim_valor,
    FIRST_VALUE(n_llibreria) OVER
        (PARTITION BY ciutat ORDER BY n_llibreria DESC
         RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) - n_llibreria AS diff
FROM
    alumne
ORDER BY
    ciutat ASC,
    n_llibreria DESC

```

Els resultats que obtenim són els següents: en aquest cas, la columna *Primer valor* calcula el nombre de llibres màxim per a Barcelona (la columna *Ciutat* defineix la partició). Podem veure com aquest valor es repeteix per a cada alumne a Barcelona. A la segona columna *Diferència*, veiem que es calcula la diferència entre la columna *Primer valor* i el nombre de llibres que aquest alumne té.

Ciutat	Nom / Cognoms	N. Llibres	Primer Valor	Diferència
Barcelona	Alejandra Martínez	11	11	0
Barcelona	Manuel Vázquez	10	11	1
Barcelona	Elena Rodríguez	6	11	5
Barcelona	José Pérez	4	11	7
Lleida	Víctor Anllada	7	7	0
Lleida	Wenceslao Fernández	6	7	1
Lleida	Victoria Setan	5	7	2
Lleida	Felisa Sánchez	5	7	2
Lleida	José María Llopis	5	7	2
Tarragona	Marina Rodríguez	9	9	0
Tarragona	Fernando Nadal	8	9	1
Tarragona	Victoria Suárez	6	9	3

Fixeu-vos que, al contrari que els altres exemples proposats fins ara, hem especificat a la consulta la clàusula `RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING`, que defineix el marc de la partició per a cada fila actual (`CURRENT ROW`). Encara que el resultat de la consulta per a `FIRST_VALUE` seria el mateix si en aquest cas no s'espècifiqués (ja que el marc estaria definit per `RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW`, que és el valor per defecte en el cas d'omissió per a PostgreSQL), és molt important destacar que en altres casos, si s'omet, ens proporcionaria uns resultats incorrectes.

3.4.7. Last value

Aquesta funció retorna el valor d'una expressió associat a l'última fila del marc definit a la consulta. Aquesta funció es defineix com a `last_value()` i conté el següent paràmetre:

```
last_value ( expression );
```

- **expression:** qualsevol valor a avaluar excepte funcions analítiques (per exemple, una columna d'una taula, una funció escalar, etc.).

Exemple d'utilització de last_value()

Suposem que tenim un requisit similar a l'exemple proposat per a `first_value()`, això és, calcular per a cada alumne la quantitat de llibres mínima entre tots els alumnes de cada ciutat, i calcular la diferència entre aquest valor i el valor de cadascun dels alumnes. En lloc de calcular la quantitat de llibres màxima, es requereix calcular la quantitat de llibres **mínima**. Aquesta consulta podríem crear-la de la següent manera:

```
SELECT
    ciutat,
    nom_cognoms,
    n_llibres,
    LAST_VALUE(n_llibres) OVER
        (PARTITION BY ciutat ORDER BY n_llibres DESC
         RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS ult_valor,
    n_llibres -
    LAST_VALUE(n_llibres) OVER
        (PARTITION BY ciutat ORDER BY n_llibres DESC
         RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS diff
FROM
    alumne
ORDER BY
    ciutat ASC,
    n_llibres DESC
```

Els resultats que obtenim són els següents:

Ciutat	Nom / Cognoms	N. Llibres	Últim Valor	Diferència
Barcelona	Alejandra Martínez	11	4	7
Barcelona	Manuel Vázquez	10	4	6
Barcelona	Elena Rodríguez	6	4	2
Barcelona	José Pérez	4	4	0
Lleida	Víctor Anllada	7	5	2
Lleida	Wenceslao Fernández	6	5	1
Lleida	Victoria Setan	5	5	0
Lleida	Felisa Sánchez	5	5	0
Lleida	José María Llopis	5	5	0
Tarragona	Marina Rodríguez	9	6	3
Tarragona	Fernando Nadal	8	6	2
Tarragona	Victoria Suárez	6	6	0

En aquesta funció és molt important especificar la clàusula de marc. La raó principal és perquè PostgreSQL estableix un marc per defecte `RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW`. Si no s'especifiqués aquesta clàusula, el valor d'*Últim valor* seria el **valor mínim trobat des de l'inici de la partició fins a la fila actual**. Vegem com serien els resultats en el cas que no s'especifiqués:

```

SELECT
    ciutat,
    nom_cognoms,
    n_llibreria,
    LAST_VALUE(n_llibreria) OVER
        (PARTITION BY ciutat ORDER BY n_llibreria DESC) AS ult_valor,
    n_llibreria - LAST_VALUE(n_llibreria) OVER
        (PARTITION BY ciutat ORDER BY n_llibreria DESC) AS diff
FROM
    alumne
ORDER BY
    ciutat ASC,
    n_llibreria DESC

```

Els resultats que obtenim a partir d'aquesta consulta sense definició de marc no són els que desitjàvem. Vegeu que *Últim valor* té el valor mínim trobat per a cada ciutat **des de l'inici de la partició fins a la fila actual, i no fins a la fi de la partició**.

Ciutat	Nom / Cognoms	N. Llibres	Últim Valor	Diferència
Barcelona	Manuel Vázquez	10	10	0
Barcelona	Elena Rodríguez	6	6	0
Barcelona	José Pérez	4	4	0
Lleida	Víctor Anllada	7	7	0
Lleida	Wenceslao Fernández	6	6	0
Lleida	Victoria Setan	5	5	0
Lleida	Felisa Sánchez	5	5	0
Lleida	José María Llopis	5	5	0
Tarragona	Marina Rodríguez	9	9	0
Tarragona	Fernando Nadal	8	8	0
Tarragona	Victoria Suarez	6	6	0

3.5. Ús de funcions d'agregació com a funcions analítiques

A més de les funcions analítiques explicades anteriorment, PostgreSQL permet la utilització de funcions d'agregació (com SUM, AVG o COUNT, entre altres) com a funcions analítiques, tal com s'ha vist en algun dels exemples presentats. En aquests casos, la funció s'encarrega d'agregar les files dins del marc definit a la consulta.

Mitjançant l'ús de funcions d'agregació com a funcions analítiques podem solucionar problemes amb alta complexitat de càcul de manera més simple. Per exemple, podem proporcionar solució a la necessitat de fer **sunes acumulatives**. Vegem-ne un exemple per a aclarir aquest cas.

Exemple de suma acumulativa utilitzant SUM com a funció analítica

Suposem que volem obtenir un llistat de noms d'alumnes, el nombre d'assignatures en les quals s'han matriculat i la suma acumulativa de les assignatures de cada alumne, ordenada per nom d'alumne. Aquesta consulta es crearia de la següent manera:

```
SELECT
    nom_cognoms,
    n_assignatures,
    SUM(n_assignatures) OVER (ORDER BY nom_cognoms ASC
        RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS suma_acumulativa
FROM
    alumne
ORDER BY
    nom_cognoms ASC
```

Els resultats que obtenim són els següents. Vegeu que el que s'obté és la suma dels elements dins del marc definit, que engloba des de la primera fila de la partició (que és tota la taula) fins a la fila actual que es processa.

Nom / Cognoms	N. Assignatures	Suma Acumulativa
Alejandra Martinez	4	4
Elena Rodriguez	3	7
Felisa Sanchez	2	9
Fernando Nadal	5	14
José María Llopis	4	18
José Pérez	9	27
Manuel Vázquez	7	34
Marina Rodriguez	4	38
Víctor Anllada	7	45
Victoria Setan	2	47
Victoria Suárez	8	55
Wenceslao Fernández	1	56

En aquests casos, hem d'anar amb compte de definir el marc correctament. Si el marc s'omet, per defecte s'assumeix un marc RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW, que per al cas que hem vist seria el mateix. En canvi, si el marc es defineix de manera diferent, podríem obtenir resultats que no són els esperats. La següent consulta fa la suma acumulativa d'assignatures, amb la diferència que el marc s'ha definit des de l'inici fins a la fi de la partició (que és tota la taula).

```
SELECT
    nom_cognoms,
    n_assignatures,
    SUM(n_assignatures) OVER (ORDER BY nom_cognoms ASC
        RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS suma_acumulativa
FROM
    alumne
ORDER BY
    nom_cognoms ASC
```

Els resultats són totalment diferents dels que s'esperaven, ja que es calcula la suma de totes les assignatures per a tota la partició de cadascun dels alumnes:

Nom / Cognoms	N. Signatures	Suma Acumulativa
Alejandra Martínez	4	56
Elena Rodríguez	3	56
Felisa Sánchez	2	56
Fernando Nadal	5	56
José María Llopis	4	56
José Pérez	9	56
Manuel Vázquez	7	56
Marina Rodríguez	4	56
Víctor Anllada	7	56
Victoria Setán	2	56
Victoria Suárez	8	56
Wenceslao Fernández	1	56

4. Tractament de valors nuls

Un valor nul (representat en SQL pel marcador `NULL`) és la forma que els SGBD relacionals utilitzen per a indicar que no existeix informació en una columna d'una fila d'una taula. En altres paraules, és una manera de representar la falta d'informació que o no sigui aplicable o bé sigui desconeguda. Un valor nul, per tant, no té cap tipus de dades associat: no és una dada numèrica, ni una cadena de caràcters ni tampoc un tipus de dada data.

Exemple de valor nul

Suposem que tenim una taula d'empleats en una base de dades operacional de recursos humans amb la següent estructura: codi, nom, número de telèfon de l'empresa i correu elèctric de l'empresa, com es pot veure a la taula inferior.

Empleats			
Codi Empleat	Nom Empleat	Telèfon	E-mail
U0001	Alejandra Martínez	652055678	amartinez@uoc.edu
U0002	José María Llopis	NULL	jmllopis@uoc.edu
U0003	Victoria Suárez	650091123	vsuarez@uoc.edu
U0004	Victoria Setan	655112345	vsetan@uoc.edu
U0005	Víctor Anllada	NULL	NULL

Per als empleats José María Llopis i Víctor Anllada veiem que el telèfon té un valor nul. La raó per la qual José María Llopis no té un valor per a telèfon és perquè el desconeixem, és a dir, aquest usuari té un número de telèfon assignat per l'empresa (ha estat vist pel Departament parlant per telèfon), però desconeixem quin és, ja que no s'ha proporcionat aquesta informació al Departament de Recursos Humans. En el cas de Víctor Anllada, sabem que aquest usuari no té número de telèfon (confirmat pel mateix usuari), perquè les seves funcions a l'empresa no requereixen que aquest usuari tingui telèfon assignat (no aplicable) ni tampoc correu elèctric (no aplicable, també apareix amb un valor nul).

4.1. Valors nuls en BD operacionals

En les BD operacionals, els valors nuls ens permeten codificar la falta d'informació en taules, com ja s'ha comentat, bé perquè no és aplicable, bé perquè aquesta és desconeguda. L'ús de valors nuls en aquest tipus de BD pot causar-nos dos tipus de problemes:

- 1) **Problemes d'eficiència:** aquest cas podria donar-se quan hem d'accedir a files d'una taula que té columnes amb valors nuls. En aquests casos, podria ser necessari revisar el model per a generar una versió alternativa de la taula, dependent de la proporció de dades (en relació amb el total) que tinguin valors nuls.

Exemple de consulta amb possibles problemes d'eficiència

Suposem que a la taula d'empleats mostrada en la introducció d'aquesta secció s'hiafegeix una columna *Taxa assegurança*. Sabem que les assegurances privades s'apliquen només a empleats especials (directius), i d'entre tots els empleats de l'empresa només un grup molt reduït d'empleats gaudeix d'aquest privilegi. Si volem obtenir els empleats amb assegurança privada a qui l'empresa paga una taxa anual d'assegurança de més de cent euros, la consulta seria la següent:

```
SELECT * FROM Empleats WHERE taxa_asseguranca >= 100
```

La consulta haurà d'accendir a tots els empleats (que podria suposar-se que és de diversos centenars) i verificar en cadascun si compleixen la condició a fi de simplement retornar, al final de l'execució, un grup d'empleats molt reduït. En aquests casos, es podria dissenyar un model relacional en el qual separen els empleats d'una banda i els empleats amb assegurança d'una altra, cosa que millorarà l'eficiència de la nostra consulta.

2) Problemes de construcció de consultes: el fet que les consultes puguin involucrar atributs amb valors nuls ens obliga que, a l'hora de construir la consulta, hagim de parar esment al fet que el resultat que ens retorna sigui el correcte, tant en el cas que hi hagi valors nuls com en el cas que no n'hi hagi.

Exemple de consulta amb tractament de nuls

Suposem la taula d'empleats mostrada en la introducció d'aquesta secció i una segona taula amb informació sobre usuaris de xarxa mostrada a continuació.

Usuaris		
Usuari	Accés a Campus	Accés a Biblioteca
amartinez@uoc.edu	Sí	No
vsuarez@uod.edu	No	Si
vsetan@uoc.edu	Sí	Si

Volem obtenir els empleats que no són usuaris. Les consultes proposades són:

```
SELECT * FROM Empleats
WHERE email NOT IN (SELECT usuari FROM Usuaris);

SELECT * FROM Empleats e
WHERE NOT EXISTS
    (SELECT * FROM Usuaris u WHERE e.email = u.usuari);
```

La primera consulta ens retornarà l'usuari amb codi d'empleat U0002, mentre que la segona consulta ens retornarà els usuaris amb codi d'empleat U0002 i U0005. La segona consulta és la que ens proporciona els resultats correctes.

A més del que s'ha comentat, és important destacar que el fet de considerar valors nuls en un SGBD fa que puguem dir que deixem de treballar amb una **lògica binària o bivalent** (veritable/fals) i que comencem a treballar amb una **lògica ternària o trivalent** (veritable/fals/desconeugut), que requereix un tractament especial per a identificar els valors que són desconeeguts o NULL (en SQL es fa mitjançant les clàusules IS NULL/IS NOT NULL). En les següents taules podem veure com s'avaluarien les operacions lògiques AND, OR i NOT en lògica ternària en comparació amb la lògica binària.

Lògica Ternària				
A	B	A OR B	A AND B	NOT A
Veritable	Veritable	Veritable	Veritable	Fals
Veritable	NULL	Veritable	NULL	Fals
Veritable	Fals	Veritable	Fals	Fals
NULL	Veritable	Veritable	NULL	NULL
NULL	NULL	NULL	NULL	NULL
NULL	Fals	NULL	Fals	NULL
Fals	Veritable	Veritable	Fals	Veritable
Fals	NULL	NULL	Fals	Veritable
Fals	Fals	Fals	Fals	Veritable

Lògica Binària				
A	B	A OR B	A AND B	NOT A
Veritable	Veritable	Veritable	Veritable	Fals
Veritable	Fals	Veritable	Fals	Fals
Fals	Veritable	Veritable	Fals	Veritable
Fals	Fals	Fals	Fals	Veritable

4.2. Valors nuls en magatzems de dades

Igual que el que hem vist anteriorment en les BD operacionals, és possible utilitzar els valors nuls en els magatzems de dades per a codificar la falta d'informació a les taules. En canvi, atesa la naturalesa d'aquests sistemes, la problemàtica de l'ús de valors nuls en aquest tipus de BD és lleugerament diferent de l'exposada en les BD operacionals. A continuació, veurem com podem fer el tractament de nuls des de dos punts de vista: tractament de nuls en taules de dimensions i tractament de nuls en taules de fets.

4.2.1. Valors nuls en taules de dimensions

Les dimensions, en el context dels magatzems de dades, representen el punt de vista que s'utilitza en l'anàlisi de les dades. Solen emmagatzemar **informació descriptiva o qualitatativa**, com noms, codis o descripcions, i s'utilitzen per a donar un context a la informació **quantitatativa** (mètriques), la qual procedeix de les taules de fets.

És molt comú trobar-se casos en els quals certes columnes que pertanyen a dimensions no tenen dades, bé perquè aquestes no existeixen en les BD operacionals, bé simplement perquè la columna no és aplicable per a la fila en qüestió.

Exemple de dimensió i dades desconegudes/no aplicables

Un exemple de dimensió en magatzems de dades és Client. Per al client, se sol guardar l'identificador del client, el nom, l'adreça, el codi postal i la ciutat on resideix. Generalment, aquestes dades solen existir en els sistemes operacionals d'una empresa, encara que no és inusual que el codi postal no estigui emmagatzemat. Aquest és un exemple clar de dada que emmagatzemarà en origen un valor nul.

També hi ha columnes que només tenen sentit depenent de les característiques de la fila. Per exemple, si es tracta d'un client individual, el sistema operacional podria emmagatzemar informació d'estat civil. Aquest tipus de columna no té sentit si es tracta d'una corporació, per la qual cosa en aquest cas s'emmagatzemaria en origen un valor nul.

Encara que és comú la utilització de valors nuls en BD operacionals, aquests no solen ser ben rebuts pels dissenyadors de magatzems de dades. Les raons principals són les següents:

- a) Diferents SGBD tenen diferents comportaments a l'hora de tractar valors nuls en clàusules WHERE, agrupacions (GROUP BY) i restriccions d'integritat.
- b) De la mateixa manera, diferents eines de creació d'informes poden tractar els nuls de diferents maneres, sobretot a l'hora de combinar dades procedents de diferents consultes a través d'informació conformada.
- c) Consultes multifet (consultes que recuperen dades des de diferents taules de fet utilitzant dimensions conformades): la manera com es fan aquestes consultes depèn de les eines de creació d'informes. Un mecanisme molt emprat és la utilització de FULL OUTER JOIN, que permet combinar diverses suconsultes que «ataquen» una taula de fets concreta a partir de dades conformades. Assegurant valors per defecte, garantim la consistència de les dades a l'hora de mostrar-les en un informe.
- d) Evitar valors nuls en llistes de valors d'informes.
- e) Existeixen productes OLAP en el mercat que no accepten valors nuls, la qual cosa ens causaria problemes a l'hora de generar cubs de dades.

FULL OUTER JOIN

Combinació externa (*outer join*) que ens permet obtenir tots els valors d'ambdues taules.

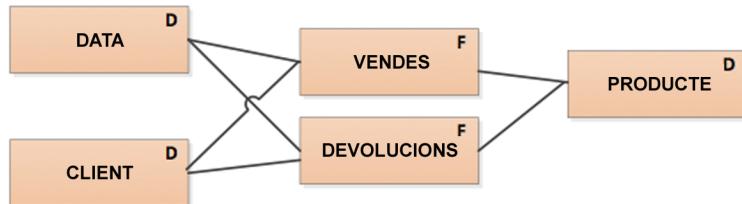
OLAP

On line analytical processing

Exemple de consulta multifet amb FULL OUTER JOIN

Suposem que tenim un model en estrella per a anàlisi d'una empresa amb Data, Client i Producte com a dimensions (D), i Ventes i Devolucions com a taules de fets (F), tal com es mostra en la figura 3.

Figura 3. Disseny en estrella proposat



Les dades de cadascuna d'aquestes taules són les següents. Les claus primàries són claus subrogades.

Data			
<u>Id Data</u>	Data	Any / Mes	Any
20150101	01-01-2015	2015/01	2015
20150102	02-01-2015	2015/01	2015
20150103	03-01-2016	2015/01	2015
20150104	04-01-2016	2015/01	2015

Client					
<u>Id Client</u>	Codi Client	Nom Client	Adreça	Codi Postal	Ciutat
1	1111	USACO S.A.	C/ Alpargata 22	31-031	Barcelona
2	2222	PENTEX S.A.	C/ Mediana 22	31-067	Tarragona
3	3333	SEMITEX S.A.	C/ Superior 102	NULL	Lleida
4	4444	FEDEX S.A.	C/ Inferior 55	30-070	Barcelona

Fixeu-vos que el codi postal del client amb clau subrogada 3 (SEMITEX S. A.) és nul.

Producte			
<u>Id Producte</u>	Codi Producte	Nom Producte	Preu Actual
1	P0001	GALETES SENSACIÓ 100gr	2.45
2	P0002	BARRA PA 250gr	1
3	P0003	POMES GALA 1 kg	1.5
4	P0004	TARONGES VALÈNCIA 1 kg	1.25
5	P0005	PLÀTANS CANÀRIES 1 kg	0.98

Vendes					
<u>Id Vendes</u>	Data	Client	Producte	N. Productes	Vendes (EUR)
1	20150101	1	1	10	24.5
2	20150101	1	2	5	5
3	20150101	2	3	3	4.5
4	20150101	2	4	1	1.25
5	20150101	3	5	NULL	14.7

Vegeu que el valor de N. productes per a la venda del client 3 (última fila) és nul. Els valors de Data, Client i Producte referencien a les claus subrogades de les taules de Data, Client i Producte respectivament.

Devolucions					
<u>Id Devolució</u>	Data	Client	Producte	N. Productes	Retornat (EUR)
1	20150103	1	1	5	12.25
2	20150103	2	2	1	1
3	20150103	3	3	1	1.5

Els valors de Data, Client i Producte referencien a les claus subrogades de les taules de Data, Client i Producte respectivament.

Imaginem que ens demanen generar la següent consulta: obtenir any/mes, nom de client, codi postal, vendes (en euros) i devolucions (en euros), ordenats per nom de client ascendentment.

Com podeu imaginar, es tracta d'una consulta multifet: d'una banda, hem d'obtenir la data, el nom del client i les vendes (emprant la taula de fets de vendes) i, d'altra banda, hem d'obtenir la data, el nom de client i les devolucions (emprant la taula de fets de devolucions). Per a facilitar l'exemple, mostrem els resultats que esperem obtenir:

Any / Mes	Nom Client	Codi Postal	Retornat (EUR)	Retornat (EUR)
2015/01	PENTEX S.A.	31-067	5.75	1
2015/01	SEMITEX S.A.	NULL	14.7	1.5
2015/01	USACO S.A.	31-031	29.5	12.25

Per a obtenir les dades de vendes farem la següent consulta:

```

SELECT
    data.any_mes,
    client.nom_client,
    client.cod_postal,
    SUM (vendes.vendes_eur) AS vendes_eur
FROM
    vendes
INNER JOIN data
ON
    vendes.data_skey = data.data_skey
INNER JOIN client
ON
    vendes.client_skey = client.client_skey
GROUP BY
    data.any_mes,
    client.nom_client,
    client.cod_postal
ORDER BY
    client.nom_client

```

Any / Mes	Nom Client	Codi Postal	Vendes (EUR)
2015/01	PENTEX S.A.	31-067	5.75
2015/01	SEMITEX S.A.	NULL	14.7
2015/01	USACO S.A.	31-031	29.5

D'altra banda, per a obtenir les dades de devolucions farem la següent consulta:

```

SELECT
    data.any_mes,
    client.nom_client,
    client.cod_postal,
    SUM (devoluciones.retornat_eur) AS retornat_eur
FROM
    devoluciones
INNER JOIN data
ON
    devoluciones.data_skey = data.data_skey
INNER JOIN client
ON
    devoluciones.client_skey = client.client_skey
GROUP BY
    data.any_mes,
    client.nom_client,
    client.cod_postal
ORDER BY
    client.nom_client

```

Any / Mes	Nom Client	Codi Postal	Retornat (EUR)
2015/01	PENTEX S.A.	31-067	1
2015/01	SEMITEX S.A.	NULL	1.5
2015/01	USACO S.A.	31-031	12.25

L'últim pas és combinar ambdues consultes en una de sola. La proposta de consulta utilitzant FULL OUTER JOIN es pot veure a continuació:

```

SELECT
    COALESCE(consulta_1.any_mes, consulta_2.any_mes) AS any_mes,
    COALESCE(consulta_1.nom_client, consulta_2.nom_client) AS nom_client,
    COALESCE(consulta_1.cod_postal, consulta_2.cod_postal) AS cod_postal,
    consulta_1.vendes_eur,
    consulta_2.retornat_eur
FROM
(
    SELECT
        data.any_mes,
        client.nom_client,
        client.cod_postal,
        SUM(vendes.vendes_eur) AS vendes_eur
    FROM
        vendes
    INNER JOIN data
        ON vendes.data_skey = data.data_skey
    INNER JOIN client
        ON vendes.client_skey = client.client_skey
    GROUP BY
        data.any_mes,
        client.nom_client,
        client.cod_postal
)
consultas_1
FULL OUTER JOIN
(
    SELECT
        data.any_mes,
        client.nom_client,
        client.cod_postal,
        SUM(devolucions.retornat_eur) AS retornat_eur
    FROM
        devolucions
    INNER JOIN data
        ON devolucions.data_skey = data.data_skey
    INNER JOIN client
        ON devolucions.client_skey = client.client_skey
    GROUP BY
        data.any_mes,
        client.nom_client,
        client.cod_postal
)
consultas_2 ON
    consulta_1.any_mes = consulta_2.any_mes
    AND consulta_1.nom_client = consulta_2.nom_client
    AND consulta_1.cod_postal = consulta_2.cod_postal
ORDER BY
    nom_client ASC

```

Si executem aquesta consulta, les dades obtingudes serien les següents, que difereixen lleugerament de les dades esperades, en concret per al client SEMITEX S. A.

Any / Mes	Nom Client	Codi Postal	Vendes (EUR)	Retornat (EUR)
2015/01	PENTEX S.A.	31-067	5.75	1
2015/01	SEMITEX S.A.	NULL	NULL	1.5
2015/01	SEMITEX S.A.	NULL	14.7	NULL
2015/01	USACO S.A.	31-031	29.5	12.25

Per què succeeix això? Veiem que a la consulta multifet, una de les condicions del **FULL OUTER JOIN** és **consulta_1.cod_postal = consulta_2.cod_postal**. En aquest cas, el codi postal és nul per a aquest client, per la qual cosa la condició d'igualtat no es compleix. El resultat de les dades per a aquest client apareix fracturat en dues files: una amb les dades de vendes i una altra amb les dades de devolucions. Vegeu també que, per a les dades de vendes, el valor de devolucions és **NULL**, i viceversa, el valor de vendes en devolucions és **NULL**. Això és així perquè no existeix informació disponible per a aquesta columna des de la taula de fets a la qual referencia.

Nota

COALESCE és una funció que retorna el primer valor no nul de la llista de valors proporcionada.

Si disposéssim de valors per defecte, els resultats obtinguts serien diferents. Assumint un valor per defecte *Desconegut* per al codi postal, els resultats obtinguts utilitzant aquesta mateixa consulta serien els següents:

Any / Mes	Nom Client	Codi Postal	Vendes (EUR)	Retornat (EUR)
2015/01	PENTEX S.A.	31-067	5.75	1
2015/01	SEMITEX S.A.	DESCONEGUT	14.7	1.5
2015/01	USACO S.A.	31-031	29.5	12.25

A partir del comentat anteriorment, la metodologia de disseny de models dimensionals recomana que **totes les columnes de les dimensions tinguin un valor per defecte**. Això és:

- Per a columnes amb tipus de dada de cadena de caràcters, se sol assignar un valor *Desconegut* o *No applicable (N/A)*, dependent de la seva naturalesa.
- Per a columnes amb tipus de dada numèrica, se sol utilitzar un valor 0.
- Per a columnes amb tipus de dada de data/hora, se sol utilitzar una data llunyana en el temps. Per exemple, 1900-01-01 o 9999-12-31, que solen indicar «des de l'inici de la història» i «fins a la fi de la història» respectivament.

Nota

La metodologia de disseny de models dimensionals és la proposada per Ralph Kimball mitjançant la seva obra titulada *The Data Warehouse Toolkit*, esmentada en la bibliografia d'aquest mòdul.

És important destacar que els valors proposats anteriorment són orientatius i que aquests solen acordar-se amb els usuaris a l'hora de dissenyar el magatzem de dades.

4.2.2. Valors nuls en taules de fets

Al contrari que les dimensions, les taules de fets contenen la informació que volem mesurar, informació que denominem **quantitativa**. Aquesta informació és la que agreguem mitjançant les funcions d'agregació (SUM, AVG, COUNT...), informació que combinem amb dimensions per a obtenir els diferents punts de vista que busquem.

Les mesures, que generalment solen ser de tipus numèric, podrien no disposar d'un valor concret (bé per motius de qualitat de dades, bé simplement perquè no era necessari guardar-lo), un escenari bastant freqüent. En aquests casos, en ser valors que seran agregats, no tindríem cap problema de permetre valors nuls. La raó per la qual això és possible és la naturalesa d'aquestes funcions, que no consideren els nuls com un valor que s'hagi de considerar. Per exemple, la suma (SUM) de valors nuls i no nuls equival a sumar solament els valors que no són nuls. El mateix succeeix amb la funció mitjana (AVG), explicar (COUNT), màxim (MAX) i mínim (MIN).

Exemple d'agregació de valors nuls

Utilitzant l'exemple anterior de vendes, podem veure que el valor de N. productes en la venda feta per al client amb clau subrogada 3 (SEMITEX S. A.) té un valor nul. Si calculem la suma, mitjana, màxim, mínim i el comptatge d'aquesta mesura per a any/mes i client, obtindrem els següents resultats.

Any / Mes	Nom Client	Suma N. Prod.	Avg. N. Prod.	Màx. N. Prod.	Min. N. Prod.	Count N. Vendes
2015/01	PENTEX S.A.	4	2	3	1	2
2015/01	SEMITEX S.A.	NULL	NULL	NULL	NULL	0
2015/01	USACO S.A.	15	7.5	10	5	2

Podem veure que totes les funcions d'agregació no tenen en compte els valors nuls a l'hora de fer els càlculs. Si haguéssim posat un valor per defecte (per exemple, 0), llavors obtindríem els següents resultats. Vegeu que per al comptatge, el valor que apareix ara és un 1, ja que existeix un valor (que és el 0 per defecte que hem posat).

Any / Mes	Nom Client	Suma N. Prod.	Avg. N. Prod.	Màx. N. Prod.	Min. N. Prod.	Count N. Vendes
2015/01	PENTEX S.A.	4	2	3	1	2
2015/01	SEMITEX S.A.	0	0	0	0	1
2015/01	USACO S.A.	15	7.5	10	5	2

Pel que fa a mètriques, és molt important entendre quina funció d'agregació s'utilitzarà per a poder decidir si és necessari utilitzar valors per defecte o no, i quin és el valor més adequat.

D'altra banda, les taules de fets contenen referències a les dimensions. Aquestes referències són les claus subrogades que hem estudiat anteriorment i que identifiquen unívocament les files de la dimensió. En ocasions, és possible trobar casos en els quals, a l'hora de mapar el valor per a obtenir la clau subrogada de la dimensió per a inserir-la en la taula de fets, no és possible trobar aquest valor (per exemple, si aquest mai no ha existit en el sistema operacional). En aquests casos, en no obtenir una clau subrogada, el valor a inserir seria un nul.

Quan ens trobem en aquest escenari, la recomanació és utilitzar un valor de clau subrogada especial per a evitar tenir valors nuls. Si recordem el vist en la secció de claus subrogades, havíem parlat d'una fila especial amb clau subrogada -1, que ens permetia identificar la falta de valors en els sistemes origen. Així, garantirem que totes les files de la taula de fets estiguin vinculades sempre a alguna fila existent en la dimensió.

Exemple de fila especial per a capturar falta de valors

Utilitzant la dimensió Client com a exemple, la fila especial amb clau subrogada -1 es podria implementar de la següent manera. Així, aconseguiríem mapar correctament les vendes i devolucions associades a clients que no podem trobar en la dimensió.

Client					
<u>Id Client</u>	Codi Client	Nom Client	Adreça	Codi Postal	Ciutat
-1	DESCONEGUT	DESCONEGUT	DESCONEGUT	DESCONEGUT	DESCONEGUT
1	1111	USACO S.A.	C/ Alpargata 22	31-031	Barcelona
2	2222	PENTEX S.A.	C/ Mediana 22	31-067	Tarragona
3	3333	SEMITEX S.A.	C/ Superior 102	NULL	Lleida
4	4444	FEDEX S.A.	C/ Inferior 55	30-070	Barcelona

Hi ha diverses raons per a utilitzar aquest mecanisme:

- Evitar violar la integritat referencial entre dimensions i fets.
- Facilitar l'ús d'`INNER JOIN` en lloc de `LEFT OUTER JOIN`, cosa que millora el rendiment de la consulta.
- Assegurar-se que els resultats agregats de la consulta són sempre correctes, independentment de la dimensió que s'utilitzi en l'anàlisi, en estar totes les dades correctament mapades.
- Facilitar la identificació de problemes de qualitat de dades. Si en fer informes es detecten valors `DESCONEGUT` o similars pot ser un senyal que hi ha problemes en el sistema operacional.

De fet, la recomanació és dissenyar la integritat referencial entre dimensions i fets amb columnes que no permetin nuls (`NOT NULL`), i forçar la utilització d'aquestes files especials en els casos descrits anteriorment.

5. Transaccions

Un dels objectius més importants dels SGBD és garantir la integritat de les dades emmagatzemades en les BD que gestionen.

La integritat té a veure amb la consistència i la qualitat de les dades. Hi ha diverses causes que poden comprometre aquesta integritat: l'accés simultani d'usuaris diferents a una mateixa BD, una situació d'avaría, el fet que s'hagi decidit tenir dades replicades per a millorar el rendiment en l'accés a la BD o que una operació pugui comprometre una regla d'integritat definida sobre la BD.

En aquest apartat veurem les possibles anomalies que es deriven de l'accés simultani de diversos usuaris a la mateixa BD i en el fet d'assegurar la disponibilitat de la BD davant de fallades o desastres, com seria el cas d'una avaria en els dispositius d'emmagatzematge extern, una apagada o un incendi.

Cal tenir present que les dades d'una organització gairebé sempre són un dels seus actius principals, una eina indispensable per al desenvolupament normal de les activitats que duu a terme.

Per tant, l'SGBD ha d'afrontar totes aquestes possibles anomalies i, per a fer-ho, es fonamenta en el concepte de transacció i en una sèrie de mecanismes per a gestionar aquestes transaccions.

5.1. Problemàtica associada a la gestió de transaccions

En els SGBD, el concepte de transacció representa la unitat de treball pel que fa a control de concorrència i recuperació. La gestió de transaccions que fa l'SGBD protegeix les aplicacions de les anomalies importants que es poden produir si no es duu a terme.

A continuació veurem, amb exemples, els problemes que poden sorgir quan s'executen de manera concurrent, i sense cap control per part de l'SGBD, diferents transaccions.

Suposem que una aplicació d'una entitat bancària ofereix als usuaris una funció que permet transferir certa quantitat de diners d'un compte d'origen a un compte de destinació.

Aquesta funció podria executar els passos que mostrem en la següent taula (en pseudocodi):

Transferència de quantitat Q de compte_origen a compte_destí	
Núm. operació	Operacions que cal executar
1	saldo_origen := llegir_saldo(compte_origen) Comprovar que saldo_origen és major o igual que Q (suposem que hi ha saldo suficient per a fer la transferència)
2	saldo_destí := llegir_saldo(compte_destí)
3	escriure_saldo(compte_origen, saldo_origen - Q)
4	escriure_saldo(compte_destí, saldo_destí + Q)
5	registrar_moviment("transferència", compte_origen, compte_destí, Q) Crear un registre per a anotar la transferència en una taula de moviments, i posar també la data i l'hora, per exemple

Cal considerar les anomalies que es produiran si no es pren cap precaució:

1) Suposem que un usuari comença a executar una d'aquestes transferències i, just després del tercer pas, una apagada fa que el procés no acabi. En aquest cas, s'haurà restat la quantitat transferida del saldo del compte d'origen, però no s'haurà sumat al del compte de destinació. Aquesta possibilitat representa un perill greu.

Des del punt de vista de l'aplicació, les operacions que s'executen quan es fa la transferència s'han de dur a terme completament o no s'han d'efectuar en absolut; és a dir, la transferència no pot quedar a mig fer.

2) Suposem que dos usuaris diferents (A i B) intenten fer dues transferències al mateix temps des de comptes d'origen diferents i cap al mateix compte de destinació. Analitzem què pot passar si, per qualsevol motiu i sense cap control per part de l'SGBD, els passos de les transaccions s'executen de manera concurrent en el següent ordre:

Execució concurrent de dues transferències bancàries			
Núm. operació	Transferència usuari A (Q = 20)	Núm. operació	Transferència usuari B (Q = 40)
1	saldo_origen:= llegir_saldo(compte_origen1) Comprovar que saldo_origen és major o igual que 20 (suposem que hi ha saldo suficient per a fer la transferència)		
2	saldo_destí:= llegir_saldo(compte_destí)		
		1	saldo_origen:= llegir_saldo(compte_origen2) Comprovar que saldo_origen és major o igual que 40 (suposem que hi ha saldo suficient per a fer la transferència)
		2	saldo_destí:= llegir_saldo(compte_destí)

Execució concurrent de dues transferències bancàries			
Núm. operació	Transferència usuari A (Q = 20)	Núm. operació	Transferència usuari B (Q = 40)
3	escriure_saldo(compte_origen1, saldo_origen - 20)		
4	escriure_saldo(compte_destí, saldo_destí + 20)		
5	registrar_moviment("transferència", compte_origen1, compte_destí, 20)		
		3	escriure_saldo(compte_origen2, saldo_origen - 40)
		4	escriure_saldo(compte_destí, saldo_destí + 40)
		5	registrar_moviment("transferència", compte_origen2, compte_destí, 40)

El resultat final és que el compte de destinació té com a saldo l'inicial més 40, en comptes de més 60. Això és incorrecte, ja que s'ha percut la quantitat que ha transferit l'usuari A.

Cal impedir d'alguna manera que l'accés concurrent de diversos usuaris produueixi resultats anòmals.

Cada usuari, individualment, ha de tenir la percepció que solament ell treballa amb la BD. En l'exemple que hem plantejat, l'execució de la transferència que efectua l'usuari B ha interferit en l'execució de la transferència que duu a terme l'usuari A. Si les dues transferències s'haguessin executat correctament aïllades l'una de l'altra, el saldo total del compte de destinació hauria estat el saldo inicial més 60.

3) Imaginem que un error de programació de la funció de transferència fa que el saldo del compte de destí rebi com a nou valor la quantitat que s'ha transferit, en comptes de sumar-la al saldo anterior. Naturalment, aquest comportament serà incorrecte, ja que no es correspon amb el desig dels usuaris, i deixarà la BD en un estat inconsistent: els saldos que haurien de tenir els comptes d'acord amb els moviments registrats (en el cinquè pas) no coincidirien amb els que s'han emmagatzemat realment.

En conclusió, és missió dels dissenyadors i programadors que les transaccions verifiquin els requisits dels usuaris.

4) Plantegem-nos què passaria si, després d'utilitzar l'aplicació durant uns quants dies i en un moment de plena activitat, es produís un error fatal del dispositiu d'emmagatzematge extern en el qual es guarda la BD, de manera que aquesta deixés d'estar disponible.

En definitiva, ha d'haver-hi mecanismes per a evitar la pèrdua tant de les dades més antigues com de les actualitzacions més recents.

5.2. Definició i propietats de les transaccions

L'accés a les dades que hi ha en una BD es fa executant operacions en l'SGBD corresponent. Com que estem interessats en SGBD relacionals, aquestes operacions, a alt nivell, seran sentències SQL. A més, per a resoldre el tipus de problemes que hem plantejat a l'apartat anterior, aquestes operacions s'agrupen en transaccions.

Una **transacció** és un conjunt d'operacions (de lectura i/o actualització) sobre la BD que s'executen com una unitat indivisible de treball. La transacció acaba la seva execució confirmant o cancel·lant els canvis que s'han dut a terme sobre la BD.

Un programa comença a treballar amb una BD connectant-s'hi d'una manera adequada i establint una sessió de treball que permet efectuar operacions de lectura i actualització (insercions, supressions, modificacions) de la BD. Per a fer una operació ha d'haver-hi una transacció activa (o en execució), que sempre és única. La transacció activa es pot iniciar mitjançant una instrucció especial o automàticament quan es fa la primera operació en l'SGBD.

Tota transacció hauria de complir quatre propietats, conegudes com a propietats ACID:

1) Atomicitat. El conjunt d'operacions que constitueixen la transacció és la unitat atòmica, indivisible, d'execució. Això vol dir que, o bé s'executen totes les operacions de la transacció (i, en aquest cas, la transacció confirma els resultats), o bé no se n'executa cap en absolut (i, en aquest cas, la transacció cancel·la els resultats). En definitiva, l'SGBD ha de garantir el tot o res per a cada transacció:

a) Per a confirmar els resultats produïts per l'execució d'una transacció, dispossem de la sentència SQL COMMIT.

b) En el cas contrari, ja sigui perquè alguna cosa impedeix que s'acabi d'executar la transacció (per exemple, un tall de llum), ja sigui perquè la transacció acaba amb una petició explícita de cancel·lació per part del programa d'aplicació, l'SGBD ha de desfer tots els canvis que la transacció hagi fet sobre la BD fins a aquest moment, com si aquesta transacció mai no hagués existit. En els dos casos es diu que la transacció ha avortat (en anglès, *abort*) l'execució. Per a cancel·lar de manera explícita els resultats produïts per l'execució d'una transacció, disposem de la sentència SQL ROLLBACK.

ACID

ACID és una sigla que es forma a partir de les inicials de les paraules atomicitat, consistència, isolament i definitivitat (*atomicity, consistency, isolation i definitivity*).

2) Consistència. L'execució d'una transacció ha de preservar la consistència de la BD. En altres paraules, si abans d'executar-se una transacció la BD es troba en un estat consistent (és a dir, en un estat en el qual es verifiquen totes les regles d'integritat definides sobre la BD), en acabar l'execució de la transacció la BD també ha de quedar en un estat consistent, si bé, mentre la transacció estigui activa, la BD podria caure momentàniament en un estat inconsistent.

3) Isolament. Una transacció no pot veure interferida la seva execució per cap altra transacció que s'estigui executant de manera concurrent amb aquesta. En definitiva, l'SGBD ha de garantir el correcte aïllament de les transaccions.

4) Definitivitat. Els resultats produïts per una transacció que confirma (és a dir, que executa l'operació de `COMMIT`) han de ser definitius en la BD; mai no es poden perdre, independentment que es produueixin fallades o desastres, fins que una altra transacció canviï aquests resultats i els confirmi. Per contra, els resultats produïts per una transacció que avorta la seva execució s'han de descartar de la BD.

És important destacar que les propietats que acabem de presentar no són independents entre si; per exemple, les propietats d'atomicitat i de definitivitat estan estretament interrelacionades. A més, el fet de garantir les propietats ACID de les transaccions no és solament una missió de l'SGBD, sinó també de les aplicacions que l'utilitzen i, per tant, del seu desenvolupador.

5.3. Interferències entre transaccions

En aquest apartat presentarem els tipus d'interferències que es poden produir si les transaccions que s'executen de manera concurrent no verifiquen la propietat d'isolament.

Abans d'entrar en aquestes interferències, és important destacar que, si hi ha dues transaccions que s'executen de manera concurrent, una d'aquestes només pot interferir en l'execució de l'altra si es donen les següents circumstàncies:

- a)** Les dues transaccions accedeixen a una mateixa porció de la BD.
- b)** Com a mínim una de les dues transaccions, sobre aquesta porció comuna de la BD a la qual accedeixen, efectua operacions d'actualització.

En altres paraules, quan les transaccions que s'executen de manera concurrent només fan lectures, no es produiran mai interferències. De manera similar, en el cas que les transaccions facin actualitzacions, si aquestes es fan sobre porcions diferents, no relacionades en la BD, tampoc no es poden produir interferències.

A continuació presentem amb exemples els tipus d'interferències que pot haver-hi entre dues transaccionis T1 i T2 que es processen de manera concurrent si no estan isolades adequadament entre si:

1) Actualització perduda. Aquesta interferència es produeix quan es perd un canvi efectuat per una transacció sobre una dada a causa de la presència d'una altra transacció que també canvia la mateixa dada. Això podria passar en una situació com la que es mostra a continuació:

Transacció T1 (reintegrament de 20)	Transacció T2 (reintegrament de 40)
saldo := llegir saldo(compte)	
	saldo := llegir saldo(compte)
escriure saldo(compte, saldo - 20)	
	escriure saldo(compte, saldo - 40)
COMMIT	
	COMMIT

T1 i T2 executen un mateix tipus de transacció; en aquest cas, un reintegrament d'un mateix compte bancari. Les dues transaccions llegeixen el mateix valor del saldo del compte, l'actualitzen independentment (assumim que hi ha saldo suficient en el compte per a fer els reintegraments) i resten d'aquest saldo la quantitat que s'ha sostret.

Suposant que l'SGBD executa les operacions que constitueixen cada transacció sense cap control i en l'ordre que es proposa en l'exemple, el canvi corresponent a la substracció de T1 es perd. En conseqüència, el saldo disminueix només en 40, en comptes de fer-ho en 60.

En definitiva, T1 ha vist interferida la seva execució a causa de la presència de T2. Si l'ordre d'execució de les operacions de cada transacció hagués estat el següent:

Transacció T1 (reintegrament de 20)	Transacció T2 (reintegrament de 40)
saldo := llegir saldo(compte)	
	saldo := llegir saldo(compte)
	escriure saldo(compte, saldo - 40)

Transacció T1 (reintegrament de 20)	Transacció T2 (reintegrament de 40)
escriure_saldo(compte, saldo - 20)	
COMMIT	
	COMMIT

s'hauria produït igualment la interferència. En aquest cas, s'hauria perdut el canvi efectuat per T2. En conseqüència, el saldo disminuiria només en 20, en comptes de fer-ho en 60. En aquest cas, T2 hauria vist interferida la seva execució a causa de la presència de T1.

En definitiva, la interferència ocorre perquè es produeixen dues lectures consecutives d'una mateixa dada (el saldo d'un mateix compte) seguides de dos canvis consecutius de la mateixa dada (de nou, el saldo d'un mateix compte). Simplement, si la seqüència d'operacions hagués estat, per exemple, la que es mostra a continuació:

Transacció T1 (reintegrament de 20)	Transacció T2 (reintegrament de 40)
saldo := llegir_saldo(compte)	
escriure_saldo(compte, saldo - 20)	
	saldo := llegir_saldo(compte)
	escriure_saldo(compte, saldo - 40)
COMMIT	
	COMMIT

la interferència no s'hauria produït. En aquest cas, T2 recuperaria el valor del saldo de compte deixat per T1 i, tenint en compte aquest nou valor de saldo per al compte, efectuaria el seu propi reintegrament. En conseqüència, el saldo del compte disminuiria en 60.

2) Lectura no confirmada. Aquesta interferència es pot produir quan una transacció recupera una dada pendent de confirmació que ha estat modificada per una altra transacció que s'executa de manera concurrent amb la transacció que recupera la dada. Això podria succeir en diverses situacions, com les que es mostren a continuació:

Transacció T1 (consulta saldo)	Transacció T2 (reintegrament de 20)
	saldo := llegir_saldo(compte)
	escriure_saldo(compte, saldo - 20)
saldo := llegir_saldo(compte)	
COMMIT	

Transacció T1 (consulta saldo)	Transacció T2 (reintegrament de 20)
	ROLLBACK

Primerament, la transacció T2 llegeix el saldo del compte i el disminueix en la quantitat que es vol reintegrar. A continuació, la transacció T1 efectua una consulta de saldo del mateix compte sobre el que T2 fa el reintegrament. El valor de saldo que obté T1 està pendent de confirmar, és una dada provisional, ja que T2 encara no ha confirmat els seus resultats. Tot seguit, la transacció T1 finalitza la seva execució i confirma els resultats. Finalment, T2 cancel·la la seva execució. Aquesta cancel·lació fa que els resultats produïts per T2 siguin descartats de la BD, de manera que el saldo del compte sigui el que hi havia abans de començar l'execució de T2. En conseqüència, T1 ha recuperat un valor que oficialment mai no ha existit i ha vist interferida la seva execució per la transacció T2. Si les transaccions T1 i T2 haguessin estat aïllades correctament, T1 mai no hauria recuperat el valor provisional deixat per T2 i que finalment ha estat descartat.

En l'exemple previ, la interferència de lectura no confirmada es produeix a causa de la cancel·lació de la transacció que modifica les dades. No obstant això, la interferència es pot produir igualment en el cas que la transacció que modifica dades confirmi els resultats.

Imaginem ara que tenim dues transaccions, T1 i T2. La transacció T1 fa la consulta d'un saldo d'un compte corrent, mentre que T2 efectua un parell de reintegraments del mateix compte corrent. Suposem que l'ordre d'execució de les operacions és el que es mostra a continuació i que l'SGBD no efectua cap control sobre l'ordre d'execució d'aquestes operacions:

Transacció T1 (consulta saldo)	Transacció T2 (dos reintegraments de 50)
	saldo := llegir saldo(compte)
	escriure saldo(compte, saldo - 50)
saldo := llegir saldo(compte)	
COMMIT	
	saldo := llegir saldo(compte)
	escriure saldo(compte, saldo - 50)
	COMMIT

En aquest cas, i encara que la transacció T2 confirma els resultats, T1 veu interferida la seva execució per T2 i recupera una dada provisional pendent de confirmació. Aquesta dada correspon a un saldo provisional per al compte corrent que correspon al saldo que queda després del primer reintegrament.

3) Lectura no repetible. Aquesta interferència es produeix quan una transacció, pels motius que sigui, necessita llegir dues vegades una mateixa dada i en cada lectura recupera un valor diferent pel fet que hi ha una altra transacció que s'executa simultàniament i que efectua una modificació de la dada llegida. Això podria passar en diverses situacions, com la que es mostra a continuació:

Transacció T1 (reintegratment de 20)	Transacció T2 (consulta saldo)
	saldo := llegir_saldo(compte)
saldo := llegir_saldo(compte)	
escriure_saldo(compte, saldo - 20)	
	saldo := llegir_saldo(compte)
COMMIT	
	COMMIT

La transacció T2, que consulta dues vegades el saldo d'un mateix compte corrent, recupera en cada lectura un valor diferent pel fet que la transacció T1, entre les dues lectures, efectua un reintegratment i interfereix en l'execució de la transacció T2. Si les transaccions s'haguessin aïllat correctament entre si, T2 hauria recuperat el mateix valor per al saldo de compte corrent en les dues lectures: o bé hauria recuperat el valor que correspongués al saldo del compte abans que s'efectués el reintegratment de T1, o bé, al saldo que quedés després que s'efectués el reintegratment de T1.

4) Anàlisi inconsistent (i el cas particular de fantasma). Els tres tipus d'interferències anteriors es produeixen pel que fa a una única dada de la BD, és a dir, ocorren quan dues transaccions intenten accedir a un mateix ítem de dades i, com a mínim, una de les dues transaccions modifica aquest ítem de dades. Però també pot haver-hi interferències pel que fa a la visió que dues transaccions tenen d'un conjunt de dades que estan interrelacionades.

Això pot passar, per exemple, quan una transacció T1 llegeix unes dades mentre que una altra transacció T2 n'actualitza una part.

T1 pot obtenir un estat de les dades incorrecte, com passa amb les següents transaccions:

Transacció T1 (consulta de saldo)	Transacció T2 (transferència)
saldo2 := llegir_saldo(compte2)	
	saldo1 := llegir_saldo(compte1)
	escriure_saldo(compte1, saldo1 - 100)
saldo1 := llegir_saldo(compte1)	

Transacció T1 (consulta de saldo)	Transacció T2 (transferència)
COMMIT	
	saldo2 := llegir_saldo(compte2)
	escriure_saldo(compte2, saldo2 + 100)
	COMMIT

Els saldo que llegeix T1 no són correctes. No es corresponen ni amb els d'abans de la transferència entre els dos comptes ni amb els de després, sinó amb un estat intermedi de T2 que no s'hauria d'haver vist mai fora de l'àmbit de T2. En conseqüència, T1 ha vist interferida la seva execució per T2.

Un cas particular bastant freqüent d'aquesta interferència són els **fantasmes**. Aquesta interferència es pot produir quan una transacció llegeix un conjunt de dades relacionat i hi ha una altra transacció que dinàmicament canvia aquest conjunt de dades d'interès afegint noves dades. Bàsicament, la interferència ocorre quan es produeix la següent seqüència d'esdeveniments:

- Una transacció T1 llegeix una sèrie de dades que compleixen una condició C determinada.
- Una transacció T2 insereix noves dades que compleixen la condició C, o bé actualitza dades que no satisfeien la condició C i que ara sí que la satisfan.
- La transacció T1 torna a llegir les dades que satisfan la condició C o bé alguna informació que depèn d'aquestes dades.

La conseqüència d'això és que T1 veu interferida la seva execució per T2 i troba un fantasma, és a dir, unes dades que abans no complien la condició i que ara sí que la compleixen. O també podria passar que T1 no veiés el fantasma directament, sinó l'efecte que té en altres dades, tal com mostren els següents exemples:

Transacció T1 (llista de comptes)	Transacció T2 (creació de comptes)
Llegir tots els comptes del banc. Imaginem que només tenim tres comptes, (compte1, compte2 i compte3) mostrar dades compte1 mostrar dades compte2 mostrar dades compte3	
	crear_compte(compte4)
	saldo_inicial(compte4, 100)
	COMMIT

Transacció T1 (llista de comptes)	Transacció T2 (creació de comptes)
sumar el saldo de tots els comptes obtenim saldo compte1 + saldo compte2 + saldo compte3 + 100 (el compte4 amb saldo 100 és el fantasma)	
COMMIT	

En aquest primer exemple, el compte 4 és un fantasma des del punt de vista de T1. I a més T1 veu l'efecte que té en la suma de saldos que es produueix i que, des del seu punt de vista, dona un resultat incoherent. Si T1 hagués estat isolada correctament de la transacció T2, una vegada executada la primera consulta, mai no hauria d'haver trobat les dades corresponents al compte 4.

Finalment, el següent exemple mostra un fantasma que es produueix a conseqüència d'una actualització de dades per part de la transacció T2. Imaginem que els propietaris dels comptes 1 i 2 viuen a Barcelona; els propietaris del compte 3 resideixen a Madrid, i els titulars del compte 4, que vivien a Tarragona, notifiquen que ara residiran a Barcelona.

Transacció T1 (llista de comptes clients de Barcelona)	Transacció T2 (canvi residència)
llegir comptes clients Barcelona mostrar dades compte1 mostrar dades compte2	
	canviar_residencia(compte4, Barcelona)
sumar el saldo de tots els comptes de clients de Barcelona obtenim saldo compte1 + saldo compte2 + saldo compte4 (el compte4 és el fantasma)	
COMMIT	
	COMMIT

En l'exemple anterior, el compte 4 és novament un fantasma des del punt de vista de la transacció T1.

5.4. Nivell de concorrència

Un SGBD pot resoldre els problemes d'interferències entre transaccions que hem vist anteriorment de dues maneres:

- a) Cancel·lar automàticament (en anglès, *abort*) les transaccions problemàtiques i desfer els canvis que han pogut produir sobre la BD.

- b)** Suspendre l'execució d'una de les transaccions problemàtiques temporalment i reprendre-la quan hagi desaparegut el perill d'interferència. En alguns casos, aquesta situació també pot comportar la cancel·lació de transaccions.

Les dues solucions impliquen un cost en termes de disminució del rendiment de la BD. Precisament, quant a la gestió de transaccions, un dels objectius dels SGBD és minimitzar aquests efectes negatius.

Es denomina **nivell de concorrència** el grau d'aprofitament dels recursos de procés disponibles, segons la coincidència de l'execució de les transaccions que accedeixen de manera concurrent a la BD i es confirmen.

L'objectiu de l'SGBD és augmentar el treball efectiu (és a dir, el treball realment útil per als usuaris) efectuat per unitat de temps. Sens dubte, les transaccions que suspenen la seva execució no fan treball efectiu, i encara menys en fan les transaccions que finalment cancel·len la seva execució.

Un dels grans reptes de la gestió de transaccions és aconseguir el nivell de concorrència adequat. Això s'aconsegueix intentant que no es produeixin cancel·lacions o suspensions d'execució de les transaccions quan no és realment necessari per a impedir una interferència. Malauradament, aquest objectiu mai no se satisfà del tot, ja que implicaria un esforç excessiu i seria perjudicial per al rendiment global per altres motius. Els SGBD intenten obtenir un compromís òptim entre el nivell de concorrència que permeten i el cost que això comporta en termes de tasques de control.

5.5. Visió externa de les transaccions

SQL estàndard força que, una vegada s'hagi establert una connexió amb la BD, la primera sentència SQL que vulguem executar mitjançant SQL interactiu **implícitament** iniciï l'execució d'una transacció. Una vegada iniciada la transacció, aquesta romandrà activa fins que **explícitament** i d'una manera obligatòria n'indiquem la finalització.

Última versió de SQL estàndard

Quan parlem de les sentències SQL, sempre ens referirem a l'última versió d'SQL estàndard, ja que té com a subconjunt totes les anteriors i, per tant, tot el que era vàlid en l'anterior ho continuarà essent en la següent. Només especificarem l'any d'una versió d'SQL quan vulguem destacar que es va fer una aportació determinada concretament en aquesta versió.

Per defecte, SQL estàndard força que aquesta transacció no vegi mai interferida la seva execució i que tampoc no pugui interferir en l'execució d'altres transaccions. En definitiva, per defecte, l'SGBD haurà de garantir el correcte isolament de totes les transaccions que accedeixin de manera concurrent a la BD.

Per a informar sobre les característiques associades a una transacció des d'SQL:1992, disposem de la següent sentència:

```
SET TRANSACTION mode_acces;
```

en la qual `mode_acces` pot ser `READ ONLY`, en el cas que la transacció només consulti la BD, o `READ WRITE`, en el cas que la transacció modifiqui la BD.

La sentència prèvia solament es pot executar en el cas que no hi hagi cap transacció en execució en la sessió de treball establerta amb la BD; si n'hi ha alguna, SQL estàndard especifica que l'SGBD hauria de reportar una situació d'error. Addicionalment, les característiques especificades seran aplicables a la resta de transaccions que s'executin posteriorment durant la sessió de treball.

Per a indicar la finalització d'una transacció, SQL estàndard ens ofereix la sentència següent:

```
{COMMIT | ROLLBACK} [WORK];
```

Mentre que `COMMIT` confirma tots els canvis produïts contra la BD durant l'execució de la transacció, `ROLLBACK` els desfà i deixa la BD com estava abans que s'iniciés la transacció. La paraula reservada `WORK` només serveix per a explicar què fa la sentència i és opcional.

Exemples d'ús de la sentència SET TRANSACTION

Suposem que tenim una BD d'un banc que guarda dades dels comptes dels clients. En concret, considerem que tenim la següent taula (clau primària subratllada):

```
comptes (num_compte, tipus_compte, saldo, comissió)
```

Considerant que hem establert la connexió amb la BD, podem executar les següents sentències durant la nostra sessió de treball amb els efectes que es comenten:

Sentència	Comentaris
<code>SET TRANSACTION READ WRITE;</code>	Informem que les transaccions que s'executaran en la sessió establerta poden fer lectures i canvis en la BD.
<code>UPDATE comptes SET saldo=saldo*1.10 WHERE num_compte="234509876";</code>	S'inicia l'execució d'una transacció que pot fer lectures i canvis en la BD. Incrementem en un 10% el saldo del compte número 234509876.

Sentència	Comentaris
SELECT * FROM comptes WHERE num_compte="234509876";	Recuperem les dades del compte número 234509876.
COMMIT;	Confirmem els resultats produïts per la transacció.
UPDATE comptes SET saldo=saldo-500 WHERE num_compte="234509876";	Aquesta sentència inicia implícitament l'execució d'una nova transacció que pot fer lectures i canvis en la BD. Transferim 500 € del compte 234509876 al compte 987656574. Suposem que hi ha suficient saldo. Disminuïm el saldo del compte d'origen.
UPDATE comptes SET saldo=saldo+500 WHERE num_compte="987656574";	Incrementem el saldo del compte de destinació.
COMMIT;	Confirmem els resultats produïts per la transacció.
SELECT saldo FROM comptes WHERE tipus_compte="estalvi a termini";	Aquesta sentència inicia implícitament l'execució d'una nova transacció que pot fer lectures i canvis en la BD. Consultem el saldo dels comptes d'un tipus determinat.
SET TRANSACTION READ ONLY;	Aquesta sentència genera un error que ens serà reportat per l'SGBD. No podem canviar les característiques de les transaccions perquè ja tenim una transacció en execució.
ROLLBACK;	Com que s'ha produït un error, cancel·lem la transacció.
SET TRANSACTION READ ONLY;	Informem que les transaccions que s'executin en la sessió de treball a partir d'aquest moment només llegiran la BD. A més, la sentència també inicia l'execució d'una transacció.
SELECT saldo FROM comptes WHERE tipus_compte="estalvi a termini";	Consultem el saldo dels comptes d'un tipus determinat.
COMMIT;	Confirmem els resultats produïts per la transacció.

El començament implícit de transaccions, en un entorn d'aplicació real, pot crear confusions sobre l'abast de cada transacció, si aquest abast no es documenta correctament. Per això, des d'SQL:1999 es proposa utilitzar la següent sentència:

```
START TRANSACTION (mode_acces);
```

en la qual mode_acces pot ser READ ONLY o READ WRITE. Si no s'especifica el mode d'accés, la sentència simplement inicia l'execució d'una nova transacció, d'acord amb les característiques que s'hagin especificat prèviament. Si abans no s'ha especificat cap característica, SQL estàndard enuncia que la transacció s'ha de considerar de tipus READ WRITE.

Inici de les transaccions

Molts SGBD incorporen sentències pròpies per a marcar explícitament l'inici de les transaccions. En la majoria dels casos, aquesta sentència és BEGIN WORK o, simplement, BEGIN, perquè la paraula clau WORK és opcional.

Exemples d'ús de la sentència START TRANSACTION

En la BD de l'exemple anterior, i assumint que hem establert la connexió amb la BD, podem executar les següents sentències amb els efectes que es comenten:

Sentència	Comentaris
START TRANSACTION READ ONLY;	Informem que comença l'execució d'una transacció de només lectura.
SELECT saldo FROM comptes WHERE tipus_compte="estalvi a termini";	Consultem el saldo dels comptes d'un tipus determinat.
COMMIT;	Confirmem els resultats produïts per la transacció.
START TRANSACTION READ WRITE;	S'inicia l'execució d'una transacció que pot consultar i modificar la BD.
UPDATE comptes SET saldo=saldo*1.10 WHERE num_compte="234509876";	Incrementem en un 10% el saldo del compte número 234509876.
SELECT * FROM comptes WHERE num_compte="234509876";	Recuperem les dades del compte número 234509876.
COMMIT;	Confirmem els resultats produïts per la transacció.
START TRANSACTION;	Inici d'una nova transacció. No se n'indiquen les característiques. Per tant, s'apliquen les especificades anteriorment. En conseqüència, la transacció pot consultar la BD i modificar-la.
UPDATE comptes SET saldo=saldo-500 WHERE num_compte="234509876";	Transferència bancària. Disminuïm el saldo del compte d'origen.
UPDATE comptes SET saldo=saldo+500 WHERE num_compte="987656574";	Incrementem el saldo del compte de destinació.
COMMIT;	Confirmem els resultats produïts per la transacció.
SELECT saldo FROM comptes WHERE tipus_compte="estalvi a termini";	Aquesta sentència inicia implícitament l'execució d'una nova transacció que pot fer lectures i canvis en la BD. Per tant, no és obligatori marcar explícitament l'inici de les transaccions, encara que sigui convenient. D'aquesta manera s'assegura la compatibilitat amb les versions prèvies de l'estàndard. Consultem el saldo dels comptes d'un tipus determinat.
COMMIT;	Confirmem els resultats produïts per la transacció.

5.5.1. Relaxació del nivell d'isolament

Fins ara havíem considerat que sempre era necessari garantir una protecció total davant de qualsevol tipus d'interferències. No obstant això, aquesta protecció total exigeix una sobrecàrrega de l'SGBD en termes de gestió d'informació de control i una disminució del nivell de concurrència.

En determinades circumstàncies, és convenient relaxar el nivell d'isolament i possibilitar que es produixin interferències. Això és correcte si se sap que aquestes interferències no es produiran realment o si en l'entorn d'aplicació en el qual ens trobem no és important que es produixin.

Si ens centrem en SQL estàndard, les instruccions `SET TRANSACTION` i `START TRANSACTION` permeten relaxar el nivell d'isolament. Tenen la següent sintaxi:

```
SET TRANSACTION {READ ONLY | READ WRITE},
ISOLATION LEVEL nivell_isolament ;

START TRANSACTION [{READ ONLY | READ WRITE}],
ISOLATION LEVEL nivell_isolament ;
```

en la qual `nivell_isolament` pot ser `READ UNCOMMITTED`, `READ COMMITTED`, `REPEATABLE READ` o `SERIALIZABLE`.

El **nivell d'isolament** determina les interferències que poden desencadenar altres transaccions en la transacció que comença. D'acord amb els tipus d'interferència que hem descrit, la següent taula indica les que s'eviten amb cada nivell d'isolament:

	Actualització perduda	Lectura no confirmada	Lectura no repetible i ànalisi inconsistent (excepte fantasma)	Fantasma
<code>READ UNCOMMITTED</code>	Sí	No	No	No
<code>READ COMMITTED</code>	Sí	Sí	No	No
<code>REPEATABLE READ</code>	Sí	Sí	Sí	No
<code>SERIALIZABLE</code>	Sí	Sí	Sí	Sí

Els nivells hi apareixen de menys a més estrictes i, per tant, de menys a més eficients:

- 1) El nivell `READ UNCOMMITTED` protegeix les dades actualitzades i evita que cap altra transacció les actualitzi fins que no s'acabi la transacció. No ofereix cap garantia pel que fa a les dades que llegeixi la transacció. Poden ser dades actualitzades per una transacció que encara no ha confirmat i, a més, una altra transacció les pot actualitzar immediatament.

2) El nivell `READ COMMITTED` protegeix parcialment les lectures i impedeix que la transacció llegeixi les dades actualitzades per una altra transacció que encara no s'hagin confirmat.

3) El nivell `REPEATABLE READ` impedeix que una altra transacció actualitzi una dada que hagi llegit la transacció fins que aquesta no s'acabi. D'aquesta manera, la transacció pot tornar a llegir aquesta dada sense risc que l'hagin canviat.

4) El nivell `SERIALIZABLE` ofereix un isolament total i evita qualsevol tipus d'interferències, inclosos els fantasmes. Això significa que no solament protegeix les dades que hagi vist la transacció, sinó també qualsevol informació de control que s'hagi utilitzat per a fer cerques.

La definició de l'SQL estàndard estableix que un SGBD concret té l'obligació de garantir com a mínim el nivell d'isolament que la transacció hagi sol·licitat, encara que pot optar per oferir un isolament més restrictiu. Per tant, l'únic nivell que un SGBD té l'obligació d'implementar és el més alt, el `SERIALIZABLE`.

5.5.2. Responsabilitats de l'SGBD i del desenvolupador

Hem vist què és una transacció i quines propietats ha de complir. Examinem la contribució de l'SGBD per a aconseguir garantir aquestes propietats i els aspectes que depenen del desenvolupador de les aplicacions.

1) Responsabilitats de l'SGBD

a) Aconseguir que l'horari que es produeixi a mesura que l'SGBD rep peticions de lectura o escriptura, i de `COMMIT` o `ROLLBACK` de les transaccions que s'executin de manera concurrent sobre la BD, sigui correcte (sense interferències). Naturalment, en el cas que s'hagi relaxat el nivell d'isolament per a algunes transaccions, serà necessari que l'SGBD consideri correctes més horaris.

L'SGBD aconsegueix horaris lliures d'interferències, sobretot, de dues maneres (no necessàriament excloents entre si): cancel·lant automàticament les transaccions problemàtiques o suspenent l'execució de la transacció fins que la pugui prendre sense problemes. El conjunt de mecanismes que es responsabilitza d'aquestes tasques es diu **control de concorrència**.

És necessari que aquests mecanismes siguin tan transparents a la programació com sigui possible, de manera que no s'afegeixin dificultats innecessàries al desenvolupament. No obstant això, de vegades és necessari oferir serveis¹ que modifiquin el comportament per defecte de l'SGBD per a augmentar el nivell de concorrència.

Horari

L'execució concurrent d'un conjunt de transaccions (en les quals es preserva l'ordre de les operacions dins de cada transacció) rep el nom d'**horari o història**.

⁽¹⁾Per exemple, la possibilitat de relaxar el nivell d'isolament de les transaccions.

- b)** Comprovar que els canvis que ha fet una transacció verifiquen totes les regles d'integritat que s'han definit en la BD. Això es pot fer just abans d'acceptar el `COMMIT` de la transacció, i rebutjar-lo si es viola alguna regla, o immediatament després que s'executi cada petició dins de la transacció.
- c)** Impedir que en la BD romanguin canvis de transaccions que no s'arriben a confirmar i que es perdin els canvis que han dut a terme transaccions confirmades en el cas que es produixin cancel·lacions de transaccions, caigudes de l'SGBD o de les aplicacions, desastres (com incendis) o fallades dels dispositius externs d'emmagatzematge. En general, parlem de **recuperació** per a referir-nos al conjunt de mecanismes que s'encarreguen d'aquestes tasques.

2) Tasques del desenvolupador d'applicacions

- a)** Identificar amb precisió les transaccions d'una aplicació, és a dir, el conjunt d'operacions que necessàriament s'ha d'executar d'una manera atòmica sobre la BD d'acord amb els requeriments dels usuaris.

En aquest sentit, les transaccions haurien de durar el mínim imprescindible. En concret, pot ser molt perillós que una aplicació tingui una transacció en execució mentre s'espera l'entrada d'informació per part de l'usuari. De vegades, els usuaris poden trigar força estona a proporcionar certes dades o, simplement, a prémer el botó d'acceptació d'un missatge. Fins i tot és possible que qualsevol circumstància els faci deixar a mig fer el que feien i que l'aplicació es quedi bastant temps esperant que l'usuari hi torni. Fins que l'usuari no permet que la transacció s'acabi, aquesta pot impedir l'actualització o fins i tot la lectura de les dades a les quals ja hagi accedit. Això significa més despresa de recursos i un fre important en el nivell de concorrència possible. Per tant, i sempre que sigui possible, se sol recomanar que durant una transacció no es pari mai l'execució de l'aplicació esperant que es produueixi una actuació determinada per part de l'usuari.

- b)** Garantir que les transaccions mantenen la consistència de la BD d'acord amb els requeriments dels usuaris i tenint en compte les restriccions d'integritat i els disparadors definits en la BD.

- c)** Considerar aspectes de rendiment. En particular, el desenvolupador ha de ser capaç d'estudiar i millorar el nivell de concorrència d'acord amb els coneixements que tingui dels mecanismes de control de concorrència de l'SGBD i les possibilitats de modificar el seu funcionament.

5.6. Transaccions a PostgreSQL

Per defecte, i si no s'indica expressament el contrari, PostgreSQL treballa amb transaccions implícites (aquest mode de treball també es coneix amb el nom d'*autocommit* activat). Això vol dir que qualsevol grup de sentències SQL que seleccionem (per exemple, des del PgAdmin) i enviem a executar serà tractat

com una transacció. Si el grup de sentències enviat no genera cap error, els resultats passaran a ser definitius en la BD. En cas contrari, els resultats seran descartats per l'SGBD.

Ja sabem que treballar amb transaccions implícites en un entorn d'aplicació real pot crear confusions sobre l'abast de cada transacció. Per això, PostgreSQL ens ofereix la sentència d'SQL estàndard `START TRANSACTION`, i també una sentència pròpia, la sentència `BEGIN`, per a indicar explícitament el començament d'una transacció. Quan s'indica explícitament el començament d'una transacció, PostgreSQL desactiva la modalitat *autocommit* i la transacció romanerà activa fins que en confirmem o en cancel·lem els resultats de manera explícita. Per a indicar la finalització de la transacció, disposem de les sentències d'SQL estàndard `COMMIT` i `ROLLBACK`.

Addicionalment, també tenim disponible la sentència `SET TRANSACTION` d'SQL estàndard per a indicar les característiques de la transacció (si és `READ ONLY` o `READ WRITE` i el nivell d'isolament) en el cas que no s'hagi fet anteriorment; per exemple, amb les sentències `START TRANSACTION` o `BEGIN`. Si l'usuari no ha especificat cap característica per a les transaccions que vol executar, per defecte, PostgreSQL considerarà que són transaccions `READ WRITE` que treballen amb un nivell d'isolament `READ COMMITTED`.

Encara que PostgreSQL permet especificar qualsevol dels nivells d'isolament proposats per SQL estàndard (`READ UNCOMMITTED`, `READ COMMITTED`, `REPEATABLE READ` i `SERIALIZABLE`), de fet, internament només treballa amb dos nivells d'isolament. Aquests nivells són els de `READ COMMITTED` i `SERIALIZABLE`:

- 1)** El nivell d'isolament `READ COMMITTED` evita que la transacció es vegi involucrada en interferències d'actualització perduda i de lectura no confirmada. La transacció es podria veure implicada en interferències de lectura no repetible i d'anàlisi inconsistent (incisos fantasmares).
- 2)** Per la seva banda, el nivell d'isolament `SERIALIZABLE` evita qualsevol tipus d'interferència.

El fet que PostgreSQL només hagi de considerar internament dos nivells d'isolament està relacionat amb el mecanisme per al control de concurrència que implementa. Aquest mecanisme es basa en el que es coneix com a model de control de concurrència multiversió (en anglès, *multiversion concurrency control*, abreujat MVCC).

Nota

Si indiquem un nivell d'isolament `READ UNCOMMITTED`, PostgreSQL, internament, el transformarà en `READ COMMITTED`.

Si indiquem un nivell d'isolament `REPEATABLE READ`, PostgreSQL, internament, el transformarà en `SERIALIZABLE`.

MVCC

El model de control de concorrència MVCC no solament es troba disponible a PostgreSQL, sinó que també ho implementen altres SGBD, com per exemple, Oracle i SQL Server.

Per a més informació sobre el model MVCC de PostgreSQL, visiteu el següent enllaç:

<http://www.postgresql.org/docs/9.3/static/mvcc.html>

5.7. Importància de les transaccions en OLTP enfront d'OLAP

Finalment, és important destacar la importància de les transaccions en sistemes OLTP en comparació dels sistemes OLAP. Per a això, utilitzarem com a base la següent taula amb les principals diferències entre aquests dos sistemes.

	OLTP	OLAP
Ús	Específic de l'aplicació	Suport a la decisió
Càrrega de treball	Predefinida	Impredictible
Accés	Lectura/Escriptura	Lectura
Complexitat de la consulta	Simple	Complexa
Files per operació	Desenes/Centenars	Milers/Milions
Nombre d'usuaris	Milers/Milions	Desenes/Centenars

Els sistemes OLTP estan concebuts per a resoldre problemes concrets (tenen un propòsit específic) i per a ser utilitzats en el dia a dia de les empreses, per la qual cosa la càrrega de treball sol estar clarament predefinida. En aquests sistemes, l'accés a les dades es fa tant per a lectura com per a escriptura (inserció de dades noves i actualitzacions de dades existents), amb una complexitat de consultes en general simple (poques taules, combinacions i agrupacions) que manegen un conjunt de dades relativament petit. Per exemple, actualitzar el nom d'un producte, actualitzar l'estoc d'un producte en concret, obtenir el llistat de productes d'una categoria en concret, etc. Finalment, el nombre d'usuaris sol ser de milers o milions.

Per tant, com ja hem vist durant aquest capítol i ateses les característiques dels entorns OLTP esmentades, l'ús de les transaccions resulta crític per a garantir la consistència de les dades i evitar així anomalies derivades de l'accés simultani d'usuaris a la mateixa BD, o bé derivades de fallades o desastres.

En contraposició, els sistemes OLAP estan pensats per a proporcionar ànalisis i donar suport a la decisió, la qual cosa significa que tenen una càrrega de treball impredictible, dependent de les necessitats de l'usuari final: per exemple, al final d'any la càrrega de treball d'aquests sistemes podria incrementar-se per la quantitat d'informació que s'ha de processar i la necessitat de, per exemple, fer informes YoY. L'accés a la informació en aquests sistemes és de lectura exclusivament, per la qual cosa un usuari final d'OLAP no farà mai actualitzacions en les dades. A més, com podrem suposar, la complexitat de les consultes a la BD és molt més gran respecte dels sistemes OLTP, ja que requereix generalment moltes agrupacions i operacions de combinació amb moltes taules, manejant grans volums de dades, com ja hem comentat, històrics. Finalment, el nombre d'usuaris, al contrari que en OLTP, sol ser d'unes desenes o centenars.

OLTP

On-line transaction processing

OLAP

On-line analytical processing

YoY

Year over year, comparació de mètriques corresponents a l'any actual respecte a l'any anterior.

Això és així perquè els usuaris finals, que generalment són analistes de negoci i directius d'empresa, són els que requereixen informació per a fer anàlisis i prendre decisions.

En conseqüència, al contrari que en OLTP, en els sistemes OLAP les transaccions no són necessàries per a garantir que els usuaris obtinguin les dades consultades i, per això, no té sentit parlar de transaccions en aquests entorns, sempre des d'un punt de vista de l'usuari final.

En canvi, és important destacar que les transaccions sí que són útils (i necessàries en molts casos) a l'hora de desenvolupar els processos ETL que proporcionen informació als sistemes OLAP. Aquests processos ETL s'encarreguen de llegir informació dels sistemes origen (que, entre altres, soLEN ser els mateixos sistemes OLTP), informació que, a través de tasques de transformació i de neteja de dades, es carrega en els sistemes OLAP. En aquests casos, una transacció ens pot resultar d'utilitat per a garantir, per exemple, l'existència de dades (recents o no) després d'un procés de càrrega de dades. Per exemple, si una taula es carrega mitjançant un procediment d'esborrament i recàrrega (és a dir, s'eliminen totes les dades de la taula i es torna a carregar de nou amb el contingut més recent), podríem necessitar una transacció per a garantir que:

- 1) Les dades antigues han estat eliminades i les noves han estat carregades, en el cas que la càrrega d'aquestes últimes hagi acabat correctament sense errors (dades carregades i confirmades mitjançant una operació de COMMIT).
- 2) En el cas que s'hagi produït un error en la càrrega de dades, les dades antigues que s'han esborrat tornin a estar disponibles per a evitar que els usuaris es quedin sense informació (cancel·lació dels canvis mitjançant una operació de ROLLBACK).

Resum

Els continguts d'aquest mòdul han estat creats amb l'objectiu de proporcionar a l'estudiant un conjunt de coneixements avançats d'SQL per al tractament de dades, orientats a entorns de magatzems de dades o *data warehouse*.

El mòdul comença presentant el concepte de clau subrogada, i n'indica l'objectiu i els beneficis que aquest tipus de mecanisme aporta. Es continua amb les *common table expression*, i es descriu la funcionalitat i l'estructura d'aquest tipus de consultes, i la possibilitat d'implementar recursivitat amb aquest mètode posant-ne un exemple pràctic. Continuem amb el concepte de funcions analítiques i la seva utilització en consultes SQL, i fem també una introducció de les funcions més importants i utilitzades. Revisem també la problemàtica dels valors nuls en BD operacionals i magatzems de dades, i les solucions a cadascun dels problemes presentats. Finalment, introduïm el concepte de transacció i les seves propietats, a més de com els SGBD implementen la gestió de transaccions per a garantir la integritat de la BD davant d'accisos simultanis de múltiples usuaris.

Tots els conceptes presentats s'han descrit teòricament, utilitzant exemples pràctics per a la seva explicació, i posteriorment s'ha indicat com aplicar-los utilitzant l'SGBD PostgreSQL. També s'afegeix l'aplicació d'aquests conceptes utilitzant l'SGBD Oracle com a annex al mòdul.

En el cas que algun estudiant vulgui aprofundir en algun dels conceptes estudiats en aquest mòdul, pot consultar les referències que hem anat indicant al llarg del mòdul o els llibres indicats a l'apartat de bibliografia.

Exercicis d'autoavaluació

1. Suposeu que tenim una dimensió Hora, en la qual cada fila representa una hora en concret amb una precisió d'hores, minuts i segons. Considereu vàlida l'opció de crear una clau subrogada amb significat en la qual el valor d'aquesta clau subrogada sigui un enter amb format HHMMSS (on HH és hora, MM minuts i SS segons)? Podeu pensar en alguna altra manera de generar una clau subrogada amb significat per a aquesta dimensió?

2. Donada la següent taula que gestiona l'històric de llocs d'empleats (Històric) amb les següents dades a dia d'avui:

Històric				
id_empleat	data_alta	nom	lloc	salari_anual
1	01-01-2015	Manuel Vázquez	Enginyer de programari	23500
2	02-01-2015	Elena Rodriguez	Analista de sistemes	16000
3	03-01-2015	José Pérez	Programador SQL	17000
1	01-03-2015	Manuel Vázquez	Cap de projectes	25500
1	01-10-2015	Manuel Vázquez	Cap de projectes sènior	26500
3	15-06-2015	José Pérez	Programador SQL sènior	19000
6	01-04-2015	Fernando Nadal	Becari	13000
2	01-10-2015	Elena Rodríguez	Responsable analistes	24500
8	01-11-2015	Victor Anllada	Cap Departament	28000

Genereu, mitjançant l'ús de CTE, la consulta SQL necessària per a obtenir el nom d'empleats existents el dia 1 de setembre de 2015, el seu lloc i salari en aquest dia, el lloc i salari actual, i la diferència entre el salari actual i el salari el dia 1 de setembre de 2015.

3. Utilitzant la taula d'Històric de llocs d'empleats de l'exercici 2, obtingueu la mateixa informació proposada en aquest exercici, afegint al resultat el rànquing d'aquests empleats sobre la base del salari anual en la data especificada, el rànquing sobre la base del salari anual a dia d'avui i, per a cada empleat, la mitjana de salari al setembre i actual per al conjunt d'empleats actius el setembre de 2015.

4. Suposem que hem dissenyat una dimensió Producte amb les següents columnes: id producte (clau subrogada, tipus enter), codi producte (tipus alfanumèric), nom producte (tipus alfanumèric), categoria producte (tipus alfanumèric), data d'alta (tipus data), data de baixa (tipus data). Suposeu que les dates d'alta i baixa s'utilitzen per a gestionar l'històric de productes. Proporcioneu el resultat final de fer les següents operacions, assumint que:

- La dimensió està buida en el moment de començar l'exercici.
- La seqüència utilitzada per a generar la clau subrogada comença amb un valor 1 i s'incrementa en 1.
- Totes les files de la dimensió han de tenir valors per a totes les columnes (no es poden inserir valors nuls).

1) S'insereix el següent producte:

Codi Producte	Nom Producte	Categoria Producte	Data d'Alta	Data de Baixa
ABCD	Galetes	NULL	01-01-2015	NULL

2) S'insereix el següent producte:

Codi Producte	Nom Producte	Categoría Producte	Data d'Alta	Data de Baixa
EFGH	Tonyina	Conserve	02-01-2015	NULL

3) S'actualitza la data de baixa del següent producte:

Codi Producte	Data de Baixa
ABCD	10-01-2015

4) S'insereix el següent producte:

Codi Producte	Nom Producte	Categoría Producte	Data d'Alta	Data de Baixa
ABCD	Pernil salat	Embotit	15-01-2015	NULL

5. Un centre mèdic privat disposa d'una BD per a guardar tota la informació sobre els seus pacients, els metges que treballen al centre, les visites mèdiques dels pacients i les receptes que es prescriuen en les visites. Entre altres, la BD inclou les dues taules que es descriuen a continuació. La clau primària de cada taula està subratllada i, tret que es digui el contrari, totes les columnes són obligatòries (no admeten valors nuls).

```
METGE (num_met, nom_metge, especialitat, adreça, ciutat, telèfon, sou, DNI)
```

Informació sobre els metges. D'aquests, se'n guarda el número de col·legiat (num_met, que és clau primària), el nom (nom_met), la seva especialitat (especialitat), l'adreça (adreça) i ciutat on viu (ciutat), telèfon (telèfon), sou (sou) i el número del document nacional d'identitat (DNI). El sou ha de ser sempre superior a zero. El DNI és únic per a cada metge, però pot ser nul si el metge no en té.

Suposem que aquesta taula conté les següents files:

METGE							
<u>num_met</u>	nom_med	especialitat	adreça	ciutat	telefon	sou	DNI
236	Pere Ba	Medicina interna	Pi 23	Lleida	678999000	3000	23456777
412	Mar Ros	Medicina interna	Born 1	Barcelona	934566549	3500	56789444

```
PACIENT (num_pac, nom_pac, adreça, ciutat, DNI)
```

Informació sobre els pacients. D'aquests, se'n guarda el número de pacient (num_pac, que és clau primària), el nom (nom_pac), l'adreça (adreça), ciutat (ciutat) i el número del document nacional d'identitat (DNI). El DNI és únic per a cada pacient, però pot ser nul si el pacient no en té.

Suposem que aquesta taula conté les següents files:

PACIENT				
<u>num_pac</u>	<u>nom_pac</u>	<u>adreça</u>	<u>ciutat</u>	<u>DNI</u>
989	Mark Smith	Or 23	Barcelona	NULL

Imaginem que sobre aquestes taules volem executar dues transaccions (T1 i T2). Aquestes transaccions volen executar les operacions que s'indiquen seguidament:

T1	T2
SELECT * FROM metge	UPDATE metge SET especialitat = 'Pediatría' WHERE num_med = 236
INSERT INTO pacient VALUES (123, 'Vicente Hijos', 'Alcalá 10', 'Madrid', '46789999')	COMMIT
SELECT * FROM metge	
COMMIT	

Suposant que l'SGBD no aplica **cap mecanisme de control de concorrència**, responeu a les següents preguntes:

- a) Proposeu, si és possible, un horari que incorpori totes les sentències SQL de T1 i T2, i que no contingui cap interferència.
- b) Proposeu, si és possible, un horari que incorpori totes les sentències SQL de T1 i T2 i que contingui una (i només una) interferència. Heu d'indicar el nom de la interferència que s'està produint i el nivell mínim d'isolament necessari d'SQL per a evitar-la.
- c) Suposeu ara que tenim una transacció T3 que s'executa de manera concurrent amb la transacció T1. El contingut de les taules és el que es mostra a l'enunciat (com si les transaccions T1 i T2 no s'haguessin executat). Proposeu, si és possible, un exemple de sentència SQL a executar per T3 que causi una interferència de tipus fantasma.

T1	T3
SELECT * FROM metge	
	SENTÈNCIA SQL
INSERT INTO pacient VALUES (123, 'Vicente Hijos', 'Alcalá 10', 'Madrid', '46789999')	
	COMMIT
SELECT * FROM metge	
COMMIT	

Solucionari

1. La solució proposada d'utilitzar un format HHMMSS com a clau subrogada d'una dimensió Hora podria ser vàlida sempre que les hores HH es representin en format 24H, és a dir, que es diferencien les hores entre les que són de matinada/demà (des de les 00:00:00 fins a les 11:59:59) i les que són de tarda (des de les 12:00:00 fins a les 23:59:59). Considerant aquest cas en concret, les claus subrogades es podrien construir de la següent manera:

Clau Subrogada	Hora Format 24H	Hora Format 12H	Hora Format AM/PM
000000	00:00:00	12:00:00	12:00:00 AM
000001	00:00:01	12:00:01	12:00:01 AM
...
120000	12:00:00	12:00:00	12:00:00 PM
120001	12:00:01	12:00:01	12:00:01 PM
120002	12:00:02	12:00:02	12:00:02 PM
...
235958	23:59:58	11:59:58	11:59:58 PM
235959	23:59:59	11:59:59	11:59:59 PM

Convé destacar que en aquest exemple, com que la clau subrogada és numèrica entera, aquelles hores de matinada (entre les 00:00:00 i les 09:59:59), que són les que comencen amb zeros, es representaran sense els zeros inicials ja que l'SGBD els elimina. Per tant, aquests casos s'emmagatzemaran en la BD de la següent manera. Aquest resultat seria el mateix que el d'utilitzar una seqüència numèrica autogenerada que comenci en zero, i assumint que les hores s'han inserit en ordre en la dimensió.

Clau Subrogada	Hora Format 24H	Hora Format 12H	Hora Format AM/PM
0	00:00:00	12:00:00	12:00:00 AM
1	00:00:01	12:00:01	12:00:01 AM
2	00:00:02	12:00:02	12:00:02 AM
...
95959	09:59:59	09:59:59	09:59:59 AM
100000	10:00:00	10:00:00	10:00:00 AM
...

Una altra manera de generar aquesta clau subrogada i continuar mantenint un significat sense eliminar aquests zeros inicials seria afegir 1000000 al valor de la clau subrogada anterior, per la qual cosa totes les hores tindran un format 1HHMMSS (1000000 seran les 00:00:00, 1150003 seran les 15:00:03, etc.).

Clau Subrogada	Hora Format 24H	Hora Format 12H	Hora Format AM/PM
1000000	00:00:00	12:00:00	12:00:00 AM
1000001	00:00:01	12:00:01	12:00:01 AM
1000002	00:00:02	12:00:02	12:00:02 AM
...
1095959	09:59:59	09:59:59	09:59:59 AM
1100000	10:00:00	10:00:00	10:00:00 AM
...

2. La consulta proposada per a obtenir els detalls específicats es pot veure a continuació:

```

WITH emp_set2015 AS (
    SELECT
        id_empleat,
        MAX ( data_alta ) AS MaxDataAlta
    FROM historic
    WHERE data_alta <= '2015-09-01'
    GROUP BY id_empleat
), emp_actual AS (
    SELECT
        id_empleat,
        MAX ( data_alta ) AS MaxDataAlta
    FROM historic
    WHERE id_empleat IN (
        SELECT
            eSet2015.id_empleat
        FROM
            emp_set2015 eSet2015
    )
    GROUP BY id_empleat
), details_set2015 AS (
    SELECT
        historic.id_empleat,
        historic.nom,
        historic.lloc,
        historic.salari_anual
    FROM historic
    INNER JOIN emp_set2015 eSet2015 ON
        eSet2015.id_empleat = historic.id_empleat
    AND eSet2015.maxdataalta = historic.data_alta
), details_actual AS (
    SELECT
        historic.id_empleat,
        historic.nom,
        historic.lloc,
        historic.salari_anual
    FROM historic
    INNER JOIN emp_actual eActual ON
        eActual.id_empleat = historic.id_empleat
    AND eActual.maxdataalta = historic.data_alta
)
SELECT
    dSet2015.nom,
    dSet2015.lloc LlocSet2015,
    dSet2015.salari_anual SalariSet2015,
    dActual.lloc LlocActual,
    dActual.salari_anual SalariActual,
    dActual.salari_anual - dSet2015.salari_anual Diferencia
FROM
    details_set2015 dSet2015
INNER JOIN details_actual dActual ON
    dSet2015.id_empleat = dActual.id_empleat;

```

3. La consulta proposada per a obtenir els detalls específicats es pot veure a continuació:

```

WITH emp_set2015 AS (
    SELECT
        id_empleat,
        MAX ( data_alta ) AS MaxDataAlta
    FROM historic
    WHERE data_alta <= '2015-09-01'
    GROUP BY id_empleat
), emp_actual AS (
    SELECT
        id_empleat,
        MAX ( data_alta ) AS MaxDataAlta
    FROM historic
    WHERE id_empleat IN (
        SELECT
            eSet2015.id_empleat
        FROM
            emp_set2015 eSet2015
    )
    GROUP BY id_empleat
), details_set2015 AS (
    SELECT
        historic.id_empleat,
        historic.nom,
        historic.lloc,
        historic.salari_anual
    FROM historic
    INNER JOIN emp_set2015 eSet2015 ON
        eSet2015.id_empleat = historic.id_empleat
    AND eSet2015.maxdataaltra = historic.data_alta
), details_actual AS (
    SELECT
        historic.id_empleat,
        historic.nom,
        historic.lloc,
        historic.salari_anual
    FROM historic
    INNER JOIN emp_actual eActual ON
        eActual.id_empleat = historic.id_empleat
    AND eActual.maxdataalta = historic.data_alta
),
SELECT
    dSet2015.nom,
    dSet2015.lloc LlocSet2015,
    dSet2015.salari_anual SalariSet2015,
    dActual.lloc LlocActual,
    dActual.salari_anual SalariActual,
    dActual.salari_anual - dSet2015.salari_anual Diferencia,
    RANK() OVER (ORDER BY dSet2015.salari_anual DESC) AS rk_salariSet2015,
    RANK() OVER (ORDER BY dActual.salari_anual DESC) AS rk_salariActual2015,
    AVG (dSet2015.salari_anual) OVER () mitjSalSet2015,
    AVG (dActual.salari_anual) OVER () mitjSalActual
FROM details_set2015 dSet2015
INNER JOIN details_actual dActual ON
    dSet2015.id_empleat = dActual.id_empleat;

```

4. El resultat de fer les operacions indicades és el següent:

- 1) Afegim el producte amb clau subrogada 1, categoria DESCONEGUT i data de baixa 31-12-9999 (indicant que no s'ha donat de baixa i que el registre és vàlid fins a la fi dels temps).

Clau Subrogada	Codi Producte	Nom Producte	Categoría Producte	Data d'Alta	Data de Baixa
1	ABCD	Galetes	DESCONEGUT	01-01-2015	31-12-9999

- 2) Afegim el producte amb clau subrogada 2 i data de baixa 31-12-9999 (indicant que no s'ha donat de baixa i que el registre és vàlid fins a la fi dels temps).

Clau Subrogada	Codi Producte	Nom Producte	Categoría Producte	Data Alta	Data Baixa
1	ABCD	Galetes	DESCONEGUT	01-01-2015	31-12-9999
2	EFGH	Tonyina	Conserves	02-01-2015	31-12-9999

3) S'actualitza el producte existent amb una nova data de baixa. Amb això aconseguim crear un període de temps en el qual la fila és vàlida (entre 01-01-2015 i 02-01-2015).

Clau Subrogada	Codi Producte	Nom Producte	Categoría Producte	Data Alta	Data Baixa
1	ABCD	Galetes	DESCONEGUT	01-01-2015	10-01-2015
2	EFGH	Tonyina	Conserves	02-01-2015	31-12-9999

4) Afegim el producte amb clau subrogada 3 i data de baixa 31-12-9999 (indicant que no s'ha donat de baixa i que el registre és vàlid fins a la fi dels temps). Veiem que el codi de producte és el mateix que l'anterior, per la qual cosa caldrà inserir-lo amb una nova clau subrogada per a mantenir l'històric de valors.

Clau Subrogada	Codi Producte	Nom Producte	Categoría Producte	Data Alta	Data Baixa
1	ABCD	Galetes	DESCONEGUT	01-01-2015	10-01-2015
2	EFGH	Tonyina	Conserves	02-01-2015	31-12-9999
3	ABCD	Pernil salat	Embotit	15-01-2015	31-12-9999

5.

a) Existeixen diverses solucions vàlides. Una de les condicions que s'han de donar perquè un horari pugui contenir interferències és que les transaccions s'executin de manera concurrent, és a dir, coincident. Per tant, per exemple, un horari en què en les transaccions no coincideixi l'execució de les sentències que incorporen no presentarà mai interferències. Tenint en compte això, una possible solució seria la següent:

T1	T2
SELECT * FROM metge	
INSERT INTO pacient VALUES (123, 'Vicente Hijos', 'Alcalá 10', 'Madrid', '46789999')	
SELECT * FROM metge	
COMMIT	
	UPDATE metge SET especialitat = 'Pediatría' WHERE num_med = 236
	COMMIT

b) Un possible exemple d'horari amb una interferència per a T1 i T2 seria el que es mostra seguidament. La interferència que es produeix és de lectura no repetible. La segona SELECT

executada per T1 recupera les mateixes files que la primera SELECT, però amb valors diferents en cada lectura (en concret, la fila 1 de la taula METGE que correspon amb el metge amb número de metge 236 és la que ha canviat el seu valor, en concret l'especialitat). El nivell mínim d'SQL que evitaria la interferència és REPEATABLE READ.

T1	T2
SELECT * FROM metge	
	UPDATE metge SET especialitat = 'Pediatría' WHERE num_med = 236
	COMMIT
INSERT INTO pacient VALUES (123, 'Vicente Hijos', 'Alcalá 10', 'Madrid', '46789999')	
SELECT * FROM metge	
COMMIT	

c) Perquè es produueixi una interferència de tipus fantasma entre T1 i T3 és necessari que T3 estengui el conjunt de dades a recuperar per les sentències SELECT que executa la transacció T1. Això podria passar amb sentències d'INSERT i UPDATE. Atès que les sentències SELECT de T1 no incorporen clàusula WHERE, l'única possibilitat seria que T3 executés una sentència d'INSERT que afegeixi un nou metge (la sentència a executar no ha de donar error). Un exemple possible seria el que es mostra seguidament:

T1	T3
SELECT * FROM metge	
	INSERT INTO metge VALUES (876, 'Ana Royo', 'Pediatría', 'Murillo', 'Sevilla', '578999111', 2700, '56111676')
INSERT INTO pacient VALUES (123, 'Vicente Hijos', 'Alcalá 10', 'Madrid', '46789999')	
	COMMIT
SELECT * FROM metge	
COMMIT	

En aquest cas, la segona SELECT de T1 recupera les dues files de la primera SELECT, més una fila extra (el fantasma) que es correspon amb el nou metge que insereix la transacció T3.

Glossari

ACID *fpl* Acrònim format per les paraules *atomicity*, *consistency*, *isolation* i *definitivity* (atomaticitat, consistència, isolament i definitivitat) que indica les propietats que tota transacció ha de complir.

BD *f* Sigla corresponent a base de dades.

base de dades operacional *f* Base de dades destinada a gestionar el dia a dia d'una organització, és a dir, emmagatzema la informació referent a l'operativa diària d'una institució.

cancel·lació d'una transacció *f* Finalització d'una transacció sense que es confirmin les actualitzacions fetes en la BD.

clau subrogada *f* Identificador únic d'una taula construït a partir d'una seqüència numèrica autogenerada i que no es deriva de les dades de l'aplicació.

common table expression *f* Tipus de consultes que utilitzen la clàusula d'SQL `WITH` amb la finalitat de simplificar i facilitar la construcció de consultes complexes.

confirmació d'una transacció *f* Finalització d'una transacció que fa que els canvis fets passin a ser definitius en la BD.

control de concorrència *m* Conjunt de tècniques que utilitza un SGBD per a evitar que es produueixin interferències entre transaccions que s'executen de manera concurrent.

CTE *f* Vegeu *common table expression*.

data warehouse *m* Vegeu magatzem de dades.

dimensió *f* Punt de vista utilitzat en l'anàlisi de cert fet.

ETL *m* Vegeu *extract, transform and load*.

extract, transform and load *m* Conjunt de processos en entorns *data warehouse* que s'encarreguen de l'extracció de dades procedents de múltiples orígens, de la transformació d'aquestes per a adequar-les a les noves estructures, i de la seva càrrega final en el magatzem de dades per al seu consum.

fet *m* Objecte d'anàlisi.

funció analítica *f* Funcionalitat proporcionada per SQL per a fer càlculs dins d'un context de manera que una fila vegi i utilitzi dades més enllà de les associades a aquesta fila.

horari *m* L'execució concurrent d'un conjunt de transaccions (en les quals es preserva l'ordre de les operacions dins de cada transacció) rep el nom d'horari o història.

interferència *f* Comportament anòmal que pot produir l'accés concurrent de diversos usuaris a la BD, si no es prenen les precaucions adequades i que posa en perill la integritat d'aquesta o fa que arribi informació errònia als usuaris.

magatzem de dades *m* Bases de dades orientades a àrees d'interès de l'empresa que integren dades de diferents fonts amb informació històrica i no volàtil i que tenen com a objectiu principal fer de suport en la presa de decisions.

mètrica *f* Dada numèrica associada a un esdeveniment que volem analitzar.

nivell d'isolament *m* Grau de protecció que ofereix l'SGBD a una transacció segons els tipus d'interferències de què la protegeix.

nivell de concorrència *m* Grau d'aprofitament dels recursos de procés disponibles segons la coincidència d'execució de les transaccions que accedeixen de manera concurrent a la BD i que aconsegueixen confirmar.

OLAP *m* Sigles que fan referència a les eines d'anàlisi, normalment multidimensional. Categoría de tecnologia de programari que permet als analistes, gestors i executius millorar el seu coneixement de les dades mitjançant l'accés ràpid, consistent i interactiu a una àmplia varietat de possibles vistes d'informació que ha estat transformada des de les dades operacionals per a reflectir la dimensionalitat real de l'empresa com l'entén l'usuari (The OLAP Council).

OLTP *m* Sigles que fan referència a sistemes operacionals, que ajuden en el dia a dia de la nostra empresa.

operació d'entrada/sortida *f* Procés que permet el transport de pàgines entre la memòria interna i la memòria externa de l'ordinador. En cada operació d'entrada/sortida (E/S) es transfereix cert nombre de pàgines. En general, per a simplificar els càlculs de cost, s'assumeix que en cada operació d'E/S es transfereix una pàgina.

operació I/S *f* Vegeu **operació d'entrada/sortida**.

pàgina *f* Unitat mínima d'accés i de transferència de dades entre la memòria interna (o principal o intermèdia) de l'ordinador i la memòria externa (no volàtil) que conté els fitxers d'una base de dades. El SO de la màquina duu a terme aquesta transferència i passa la pàgina a l'SGBD perquè en gestioni el contingut. La pàgina també és l'estructura que permet a l'SGBD organitzar les dades d'una base de dades. A l'àrea de SO la pàgina rep el nom de bloc.

partició *f* Sistema d'emmagatzematge que permet distribuir les dades d'una taula en diferents espais físics per a augmentar l'eficiència del sistema.

SGBD *m* Vegeu **sistema de gestió de bases de dades**.

sistema de gestió de bases de dades *m* Programari que gestiona i controla bases de dades. Les seves funcions principals són les de facilitar l'ús simultani a molts usuaris de diferents tipus, independitzar l'usuari de les estructures físiques que implementen la base de dades i mantenir la integritat de les dades.

sistema operacional *m* Sistema que ajuda en les operacions diàries de negoci d'una organització.

transacció *f* Conjunt d'operacions de lectura i actualització de la base de dades que acaba confirmant o cancel·lant els canvis que s'han dut a terme.

Bibliografia

- Adamson, C.** (2010). *Star Schema: The Completi Reference*. McGraw-Hill.
- Bernstein, P. A.; Newcomer, I.** (2009). *Principles of Transaction Processing* (2a. ed.). Burlington (Massachusetts): Morgan Kaufmann Publishers.
- Chauhan, C.** (2015). *PostgreSQL Cookbook*. Packt Publishing.
- Elmasri, R.; Navathe, S. B.** (2007). *Sistemas de bases de datos. Conceptos fundamentales* (5a. ed.). Madrid: Addison-Wesley Iberoamericana.
- Kimball, R.; Ross, M.** (2013). *The Data Warehouse Toolkit* (3a. ed.). John Wiley & Sons, Inc.
- Liu, L.; Özsu, M. T.** (eds.) (2009). *Encyclopedia of Database Systems*. Springer.
- Morton, K; Osborne, K.; Sands, R.; Shamsudden, R; Still, J.** (2013). *Pro Oracle SQL* (2a. ed.). Apress.
- Obe, R. O.; Hsu, L. S.** (2014). *PostgreSQL: Up and Running* (2a. ed.). O'Reilly Mitjana, Inc.
- Oracle.** Manuals accessibles en línia des del web <http://www.oracle.com/technetwork/indexes/documentation/index.html>.
- PostgreSQL.** Manuals accessibles en línia des del web <http://www.postgresql.org/docs/>.
- Prigmore, M.** (2008). *An introduction to databases with web applications*. Pearson.
- Shahzad, A.; Fayyaz, A.; Ahmed, I.** (2015). *PostgreSQL Developer's Guide*. Packt Publishing.

Annexos: complements d'SQL – anotacions per a Oracle

En els següents annexos es presenten les anotacions en SQL pertanyents a l'SGBD Oracle, en les quals es poden en relleu les diferències entre aquest SGBD i PostgreSQL. Les sentències SQL proporcionades, i també les explicacions, estan basades en els exemples proposats respectivament en cadascuna de les seccions d'aquest mòdul didàctic.

Annex 1. Claus subrogades

1) Seqüències

La definició de seqüències a Oracle es fa de la següent manera:

```
CREATE SEQUENCE sequence_name
  [ INCREMENT BY integer ]
  [ START WITH integer ]
  [ MAXVALUE integer | NOMAXVALUE ]
  [ MINVALUE integer | NOMINVALUE ]
  [ CYCLE | NOCYCLE ]
  [ CACHE # | NOCACHE ]
  [ ORDER | NOORDER ]
```

Com a exemple, definirem la seqüència que la taula Assignatura utilitzarà:

```
CREATE SEQUENCE seq_assignatura_key INCREMENT BY 1 START WITH 1 NOCYCLE
```

Per a fer ús de la seqüència a Oracle, les seqüències disposen de la funció `nextval`. Aquesta funció és similar a la mostrada a PostgreSQL, amb la diferència que no disposa de paràmetres i que aquesta s'anomena amb el nom de la seqüència a l'inici:

```
SELECT seq_assignatura_key.nextval FROM dual
```

A l'hora d'inserir una fila nova a la taula Assignatura, podem fer una crida a la funció com a part de la sentència `INSERT`:

Nota

L'explicació dels paràmetres utilitzats en la creació d'una seqüència es pot consultar a la següent URL d'Oracle 11g:
http://docs.oracle.com/cd/b28359_01/server.111/b28286/statements_6015.htm#SQLRF01314

```
INSERT INTO assignatura (assignatura_key, cod_assignatura, nom_assignatura)
VALUES (seq_assignatura_key.nextval, 'UOC-31238', 'Calcul Numeric')
```

Si volem automatitzar aquesta solució a fi d'evitar utilitzar el nom de la seqüència en cada sentència `INSERT`, podem utilitzar un disparador:

```
CREATE TRIGGER tg_assignatura_key
    BEFORE INSERT ON assignatura
    FOR EACH ROW
BEGIN
    SELECT seq_assignatura_key.nextval INTO :new.assignatura_key
    FROM dual;
END
;
INSERT INTO assignatura (cod_assignatura, nom_assignatura)
VALUES ('UOC-312999', 'Programació Java');
```

2) Ús d'IDENTITY

En versions d'Oracle 11g i inferiors, no existeix un tipus de dada o clàusula específica que permeti generar valors numèrics autoincrementals. En la seva versió 12c, Oracle ha implementat aquesta funcionalitat incorporant la clàusula `IDENTITY`, que té un funcionament similar al tipus de dada `serial` vist a PostgreSQL. Aquesta clàusula s'associa a columnes amb tipus de dades numèriques a l'hora d'especificar les sentències de creació de taules. La sintaxi d'aquesta clàusula és la següent:

```
CREATE TABLE table_name (
    col_name numeric_type GENERATED [ALWAYS | BY DEFAULT] AS IDENTITY,
    ...
);
```

Nota

Per a més informació sobre aquesta clàusula a Oracle 12c, recomanem visitar la documentació en línia, a la qual es pot accedir anant a aquesta URL:

<http://docs.oracle.com/database/121/sqlrf/toc.htm>
<http://docs.oracle.com/database/121/sqlrf/toc.htm>

Annex 2. Common table expression

1) Construcció de CTE

En el cas d'Oracle, la possibilitat de crear consultes CTE s'ha introduït en la versió Oracle 9.2, i la possibilitat de crear consultes recursives des de la versió d'Oracle 11g Release 2. La sintaxi de creació de CTE a Oracle és molt similar a la proporcionada per PostgreSQL, en la qual s'afegeixen funcionalitats específiques d'aquest SGBD.

```

WITH aliases_1 [ ( c_aliases_1, c_aliases_2, ... ] ) AS (
    query_1
)
[search_clause]
[cycle_clause]
...
    aliases_n [ ( c_aliases_1, c_aliases_2, ... ] ) AS (
        query_n
)
[search_clause]
[cycle_clause]

SELECT ...
    FROM ...
    WHERE ...

[hierarchical_query_clause]

GROUP BY ...
ORDER BY ...

```

D'una banda, podem veure que la paraula RECURSIVE no és necessària a Oracle. D'altra banda, Oracle afegeix una funcionalitat especial per a cada consulta auxiliar: `search_clause` i `cycle_clause`, que es poden veure a continuació.

La clàusula `search_clause` s'utilitza per a especificar ordenació a les files:

```

SEARCH
{ DEPTH FIRST BY c_aliases [, c_aliases]...
  [ ASC | DESC ]
  [ NULLS FIRST | NULLS LAST ]
[BREADTH FIRST BY c_aliases [, c_aliases]...
  [ ASC | DESC ]
  [ NULLS FIRST | NULLS LAST ]
}
SET ordering_column

```

Nota

Per a més informació sobre les CTE a Oracle 11g, visiteu l'enllaç següent:
http://docs.oracle.com/cd/e11882_01/server.112/e41084/statements_10002.htm#SQLRF01702

- L'ordenació s'estableix especificant les columnes (llista de `c_aliases`) a partir de la clàusula `FIRST BY`.
- La llista de columnes (`c_aliases`) han de ser noms de columnes de la llista de la consulta auxiliar (`c_aliases_1, c_aliases_2`, etc.).
- Si especificuem `BREADTH FIRST BY`, les files germanes es retornaran primer abans que les files filles. Al contrari, si s'especifica `DEPTH FIRST BY`, les files filles es retornaran primer abans que les files germanes.
- `SET ordering_column` s'utilitza per a especificar el nom de l'ordre estableert per la clàusula `SEARCH`, i que pot ser utilitzat per la consulta principal per a retornar les dades en aquest mateix ordre.

La clàusula `cycle_clause` s'utilitza per a especificar cicles de valors quan s'aplica recursitat. Bàsicament, el que s'està fent és crear una nova columna que pot ser referenciada a la consulta principal i que ens permet detectar si certs valors han estat repetits o no dins de les files ancestrals dins de la jerarquia:

```
CYCLE c_aliases [, c_aliases]...
  SET cycle_mark c_aliases TO cycle_value
  DEFAULT no_cycle_value
```

- La llista de columnes (`c_aliases`) han de ser noms de columnes de la llista de la consulta auxiliar (`c_aliases_1`, `c_aliases_2`, etc.).
- S'han d'especificar dos valors: el valor `cycle_value`, que és el valor que s'assigna a la columna generada si s'ha trobat un cicle, i el valor `no_cycle_value`, que és el valor per defecte quan no s'ha trobat un cicle. Tots dos valors han de ser una seqüència de caràcters alfanumèrics de com a mínim un caràcter.
- En el cas que es trobi un cicle per a una fila en concret, el procés de recursió per al càlcul d'aquest cicle es deté en aquesta fila, és a dir, no continuará buscant. Sí que continuará per a les files que encara no han trobat un cicle.

Exemple de consulta CTE amb SEARCH i CYCLE

Utilitzant la taula `d'empleats`, volem obtenir el llistat d'aquests (nom), al costat de l'identificador de supervisor i ciutat. Per a explicar l'ús de `SEARCH` i `CYCLE`, generarem la jerarquia en l'ordre d'aparició en la jerarquia alfabetínicament mitjançant `SEARCH`, i marcarem l'empleat en el cas que aquest visqui a la mateixa ciutat que algun dels seus supervisors mitjançant `CYCLE`. La consulta que ens proporciona aquesta informació és la següent (vegeu les parts ressaltades en negreta).

```
WITH jerarquia (id_empleat,
                 nom,
                 id_supervisor,
                 nivell_jerarquia,
                 ciutat) AS (
    SELECT
        id_empleat,
        nom,
        id_supervisor,
        0 nivell_jerarquia,
        ciutat
    FROM empleat
    WHERE id_supervisor IS NULL
    UNION ALL
    SELECT
        e.id_empleat,
        e.nom,
        e.id_supervisor,
        d.nivell_jerarquia + 1 nivell_jerarquia,
        e.ciutat
    FROM jerarquia d, empleat e
    WHERE d.id_empleat = e.id_supervisor
)
SEARCH DEPTH FIRST BY id_supervisor, nom SET ordre_1
CYCLE CIUTAT SET es_cicle TO 'Y' DEFAULT 'N'
SELECT
    id_empleat,
    lpad ('|- ', 3 * nivell_jerarquia ) || nom AS empleat,
    id_supervisor,
    ciutat,
    es_cicle
FROM jerarquia
ORDER BY ordre_1
```

Els resultats d'aquesta consulta són els següents. Podem veure que hem intentat la jerarquia capturant el nivell en el qual es troba cada empleat, i hem presentat les dades de manera que primer vam mostrar les files filles i després les files germanes (DEPTH FIRST BY) ordenades alfabèticament. Amb això el que fem és un recorregut en profunditat d'un arbre. A més, si ens fixem en els empleats Manuel Bertrán i Elena Rodríguez, veiem que s'ha indicat `es_cicle = 'Y'`. Això significa que aquests empleats viuen a la mateixa ciutat que algun dels supervisors de la seva jerarquia. En el cas de Manuel Bertrán, el seu supervisor José María Llopis viu a la mateixa ciutat, i en el cas d'Elena Rodríguez és la supervisora Victoria Suárez la que viu a la mateixa ciutat.

Id Empleat	Nom	Id Supervisor	Ciutat	Es Cicle
10	Victoria Setan		Casteldefels	N
9	- José María Llopis	10	Barcelona	N
11	- Manuel Bertrán	9	Barcelona	Y
8	- Víctor Anllada	10	Lleida	N
7	- Victoria Suárez	10	Tarragona	N
4	- Alejandra Martínez	7	Barcelona	N
6	- Fernando Nadal	4	Viladecans	N
5	- Marina Rodriguez	4	Vilanova	N
1	- Manuel Vázquez	7	Barcelona	N
2	- Elena Rodríguez	1	Tarragona	Y
3	- José Pérez	1	Girona	N

2) Consultes jeràrquiques

Oracle permet fer consultes sobre dades jeràrquiques sense utilitzar consultes CTE. A aquestes consultes, Oracle les ha denominat **consultes jeràrquiques**, i per a la seva implementació ha optat per la definició de la clàusula CONNECT BY, la sintaxi de la qual es pot veure a continuació:

```
CONNECT BY [NOCYCLE] condition [AND condition]... [START WITH condition]
| START WITH condition CONNECT BY [NOCYCLE] condition [AND condition]...
```

Consultes jeràrquiques

Per a més informació sobre les consultes jeràrquiques a Oracle 11g, visiteu l'enllaç següent:
http://docs.oracle.com/cd/e11882_01/server.112/e41084/queries003.htm#SQLRF52335

Si bé és una funcionalitat interessant per a fer consultes, aquesta està fora de l'abast d'aquest mòdul i per tant no s'hi aprofundirà.

Exemple de consulta jeràrquica

La següent consulta obté la mateixa informació que la consulta generada prèviament amb SEARCH i CYCLE, però utilitzant CONNECTBY.

```
SELECT
    id_empleat,
    LPAD ('| ', 3 * (TO_NUMBER(LEVEL) - 1)) || nom AS jerarquia,
    id_supervisor,
    ciutat
FROM
    EMPLEAT START WITH id_empleat = 10
CONNECT BY PRIOR id_empleat = id_supervisor
ORDER SIBLING BY nom
```

Id empleat	Nom	Id supervisor	Ciutat
10	Victoria Setan		Casteldefels
9	- José María Llopis	10	Barcelona
11	- Manuel Bertrán	9	Barcelona
8	- Victor Anllada	10	Lleida
7	- Victoria Suarez	10	Tarragona
4	- Alejandra Martinez	7	Barcelona
6	- Fernando Nadal	4	Viladecans
5	- Marina Rodriguez	4	Vilanova
1	- Manuel Vázquez	7	Barcelona
2	- Elena Rodriguez	1	Tarragona
3	- José Pérez	1	Girona

Annex 3. Funcions analítiques

1) Crides a funcions analítiques

La crida a funcions analítiques a Oracle és molt similar a la que s'ha proposat a PostgreSQL. Aquestes han estat afegides en la versió d'Oracle 8.i Release 2.

La sintaxi és la següent:

```
analytic_function ([ arguments ]) OVER ( analytic_clause )
```

En aquesta definició, **arguments** representa qualsevol expressió que no contingui una crida a una funció analítica: podria tractar-se d'una columna d'una taula, una funció d'agregació, una constant o un càlcul, entre altres.

La sintaxi d'**analytic_clause** és la següent:

```
[ PARTITION BY { expr[, expr ]... | ( expr[, expr ]... ) } ]
[ ORDER [ SIBLINGS ] BY
  { expr | position | c_aliases }
  [ ASC | DESC ]
  [ NULLS FIRST | NULLS LAST ]
  [, { expr | position | c_aliases }
  [ ASC | DESC ]
  [ NULLS FIRST | NULLS LAST ]
  [ frame_clause ]
]
```

en la qual **frame_clause**, que permet definir el marc de treball, es defineix com a:

```

{ ROWS | RANGE }
{ BETWEEN
    { UNBOUNDED PRECEDING | CURRENT ROW | offset { PRECEDING | FOLLOWING } }
    AND
    { UNBOUNDED FOLLOWING | CURRENT ROW | offset { PRECEDING | FOLLOWING } }

| UNBOUNDED PRECEDING

| CURRENT ROW

| offset PRECEDING
}

```

A diferència de PostgreSQL, Oracle té les següents particularitats:

- a) Permet especificar en una mateixa consulta diferents funcions analítiques amb diferents finestres mitjançant `PARTITION BY`, però no permet l'ús de la clàusula `WINDOW` com a PostgreSQL.
- b) La clàusula de definició del marc obliga a definir `ORDER BY`, és a dir, no es pot definir un marc sense definir `ORDER BY` com a part de la funció analítica.
- c) La clàusula `ORDER BY` permet només una expressió en el cas que s'usi `RANGE` i una expressió amb `offset` per a determinar els límits.
- d) *Offset* solament pot prendre valors de tipus numèric o un interval.
- e) En el cas que s'usi una expressió amb `offset`, el tipus de dada de l'expressió usada en `ORDER BY` ha de ser numèrica o de tipus data.

Com a exemple, la següent consulta no seria vàlida perquè el tipus de dada de `nom_cognoms` és una cadena de caràcters.

```

SELECT
    ciutat
    nom_cognoms,
    n_assignatures,
    SUMA (n_assignatures) OVER (PARTITION BY ciutat
        ORDER BY nom_cognoms ASC
        RANGE BETWEEN UNBOUNDED PRECEDING AND 1 FOLLOWING) AS suma
FROM
    alumne
ORDER BY
    ciutat ASC,
    nom_cognoms ASC

```

Figura 4. Error que produeix Oracle si el tipus de dada en ORDER BY és diferent a una data o nombre.

```
ORA-00902: invalid datatype
00902. 00000 - "invalid datatype"
*Cause:
*Action:
Error at Line: 6 Column: 14
```

f) En el cas que s'ometi la definició del marc, l'opció seleccionada per Oracle és RANGE UNBOUNDED PRECEDING, que és equivalent a especificar RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW.

g) Al contrari que PostgreSQL, el límit inferior del marc sí que pot prendre un valor UNBOUNDED FOLLOWING, però obliga que el límit superior sigui també UNBOUNDED FOLLOWING. Al seu torn, el límit superior sí que pot prendre un valor UNBOUNDED PRECEDING, però el límit inferior també ha de ser definit com a UNBOUNDED PRECEDING.

2) Tipus de funcions analítiques

Row number

La funció `row_number` a Oracle no accepta paràmetres.

```
row_number()
```

Nota

Consulteu el següent vincle per a obtenir una descripció de totes les funcions analítiques presents a Oracle 11g:
http://docs.oracle.com/cd/e11882_01/server.112/e41084/functions004.htm#SQLRF06174

Rank

La funció `rank` a Oracle no accepta paràmetres.

```
rank()
```

Dense_rank

La funció `dense_rank` a Oracle no accepta paràmetres.

```
dense_rank()
```

Lag

La funció `lag` a Oracle accepta els següents paràmetres:

```
lag ( expression [{RESPECT | IGNORE} NULLS] [, offset] [, default] );
```

- **expression**: qualsevol valor a avaluar excepte funcions analítiques (per exemple, una columna d'una taula, una funció escalar, etc.).
- **offset** (opcional): indica la posició de la fila prèvia a la fila a què s'accendirà des de la fila actual en la partició. Per exemple, un valor de 3 indica que s'accendirà a la tercera fila prèvia a la fila actual. Si s'omet, per defecte s'assigna un valor 1 (la fila anterior).
- **default** (opcional): el valor per defecte que s'ha d'assignar en el cas que la fila a què s'ha d'accendir estigui fora dels límits permesos. Si s'omet, per defecte s'assigna un valor NULL.

En el cas d'Oracle, existeix una funcionalitat que permet incloure o eliminar els valors nuls d'`expression` del càlcul de la funció. Això s'indica mitjançant les clàusules `RESPECT NULLS` o `IGNORE NULLS` just després de l'`expression` a avaluar. Per defecte, Oracle assumeix que s'especifica `RESPECT NULLS` en el cas d'omissió.

Lead

La funció `lead` a Oracle accepta els següents paràmetres:

```
lead ( expression [{RESPECT | IGNORE} NULLS] [, offset] [, default] );
```

- **expression**: qualsevol valor que s'ha d'avaluar excepte funcions analítiques (per exemple, una columna d'una taula, una funció escalar, etc.).
- **offset** (opcional): indica la posició de la fila posterior a la fila a què s'accendirà des de la fila actual en la partició. Per exemple, un valor de 3 indica que s'accendirà a la tercera fila posterior a la fila actual. Si s'omet, per defecte s'assigna un valor 1 (la fila següent).
- **default** (opcional): el valor per defecte a assignar en el cas que la fila a què s'ha d'accendir estigui fora dels límits permesos. Si s'omet, per defecte s'assigna un valor NULL.

En el cas d'Oracle, i igual que passa amb `lag`, és possible incloure o eliminar els valors nuls d'`expression` del càlcul de la funció mitjançant les clàusules `RESPECT NULLS` o `IGNORE NULLS` just després de l'`expression` a avaluar. Per defecte, Oracle assumeix que s'especifica `RESPECT NULLS` en el cas d'omissió.

First value

La funció `first_value` a Oracle accepta els següents paràmetres:

```
first_value ( expression [{RESPECT | IGNORE} NULLS] );
```

- **expression:** qualsevol valor a avaluar excepte funcions analítiques (per exemple, una columna d'una taula, una funció escalar, etc.).

En el cas d'Oracle, i igual que passa amb altres funcions analítiques, és possible incloure o eliminar els valors nuls d'`expression` del càlcul de la funció mitjançant les clàusules `RESPECT NULLS` o `IGNORE NULLS` just després de l'expressió a avaluar. Per defecte, Oracle assumeix que s'especifica `RESPECT NULLS` en el cas d'omissió.

Last value

La funció `last_value` a Oracle accepta els següents paràmetres:

```
last_value ( expression [{RESPECT | IGNORE} NULLS] );
```

- **expression:** qualsevol valor a avaluar excepte funcions analítiques (per exemple, una columna d'una taula, una funció escalar, etc.).

En el cas d'Oracle, i igual que passa amb altres funcions analítiques, és possible incloure o eliminar els valors nuls d'`expression` del càlcul de la funció mitjançant les clàusules `RESPECT NULLS` o `IGNORE NULLS` just després de l'expressió a avaluar. Per defecte, Oracle assumeix que s'especifica `RESPECT NULLS` en el cas d'omissió.

