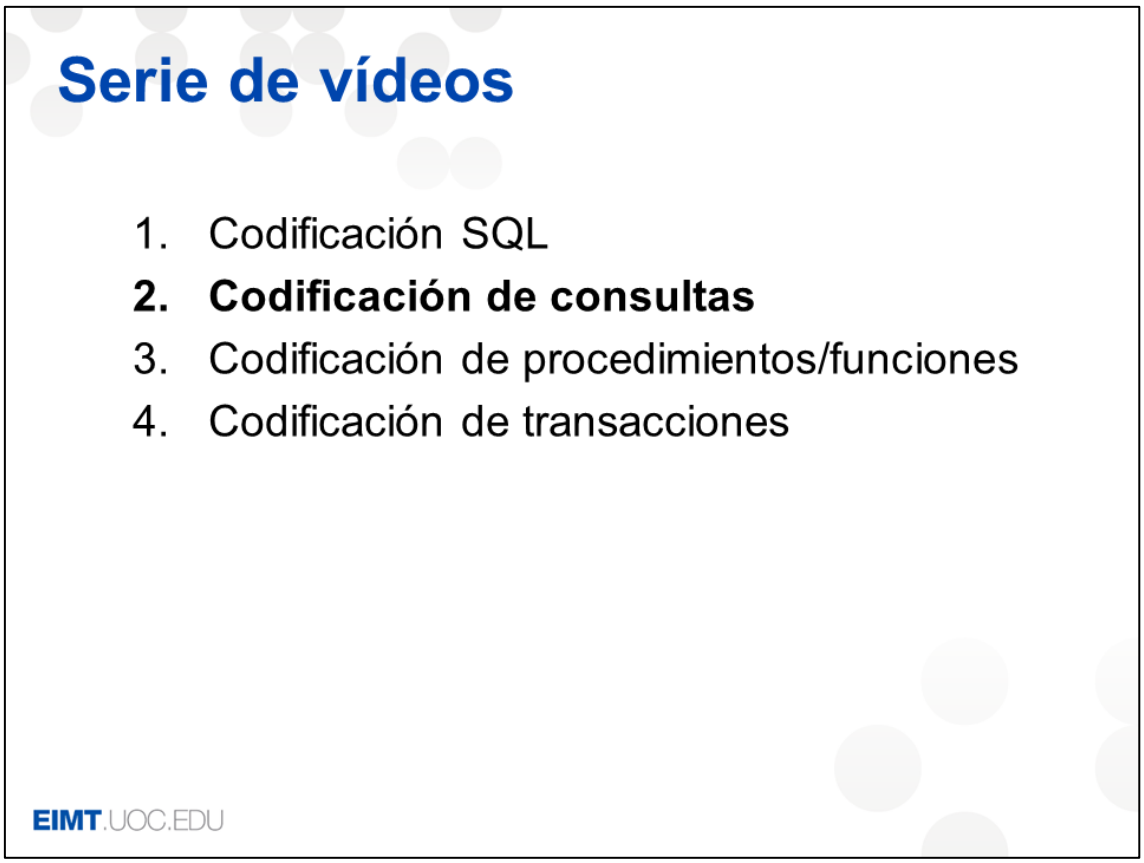
Bones pràctiques en SQL

Codificació de consultes

PID_00253043

Alexandre Pereiras Magariños

Universitat Oberta de Catalunya



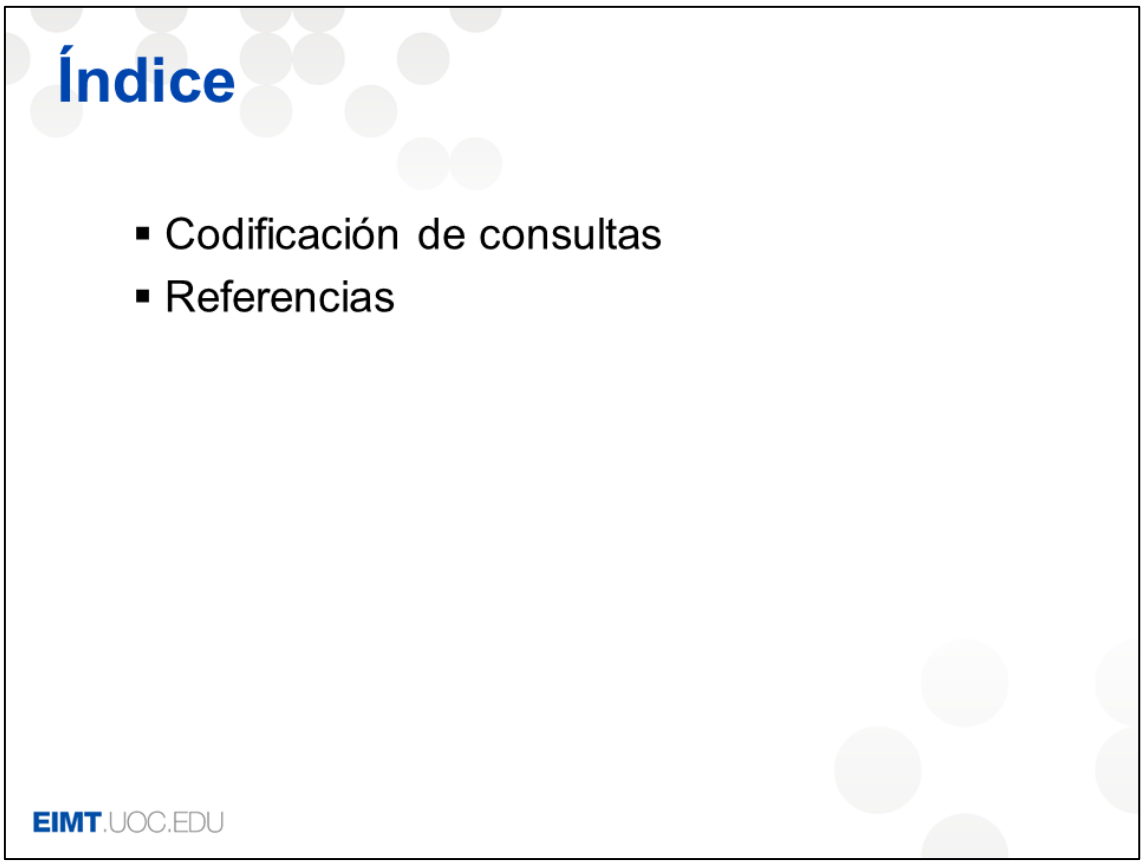
Benvinguts al segon vídeo de la sèrie *Bones pràctiques en SQL*, sèrie en què veurem un conjunt de bones pràctiques a l'hora de programar SQL en entorns de bases de dades/*data warehouse*.

Aquesta sèrie de vídeos està dividida en 4 parts:

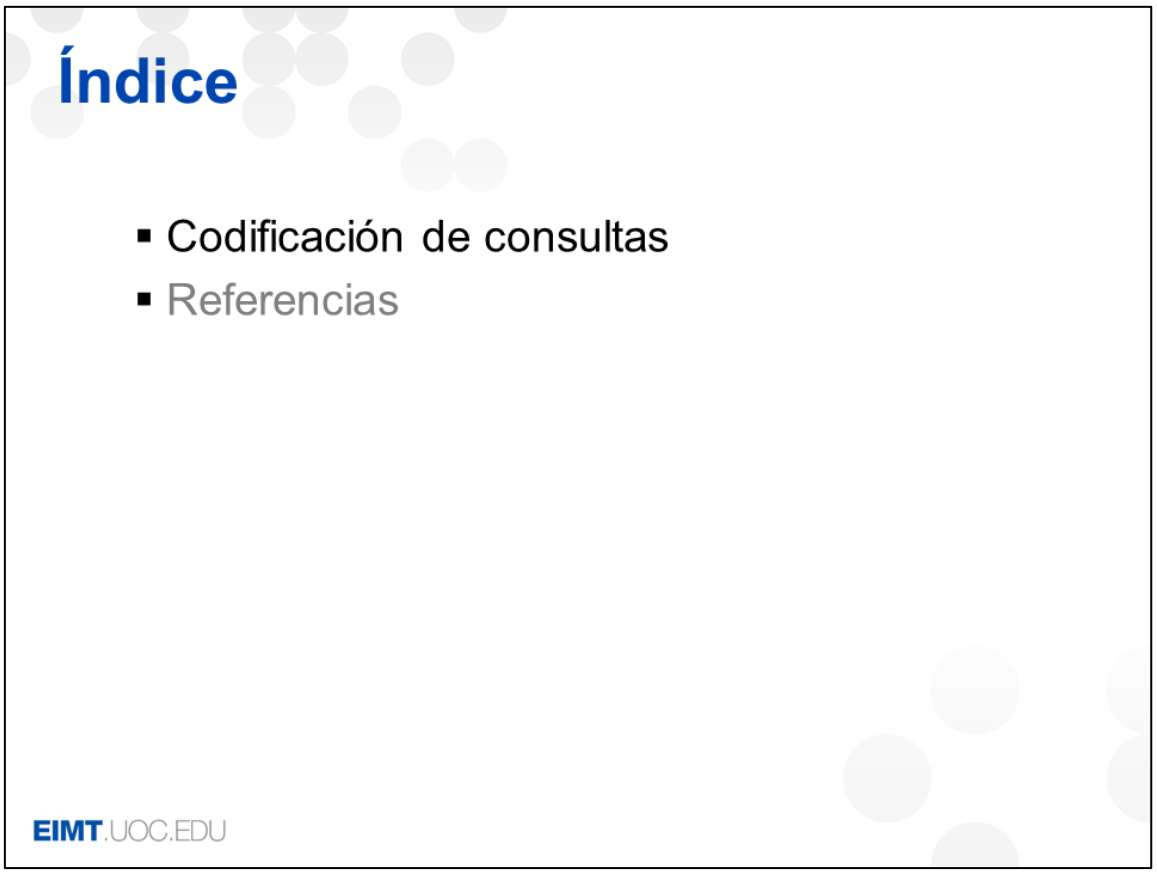
- *Codificació SQL*: centrada en aquelles pràctiques des del punt de vista genèric en SQL, que afecten la llegibilitat i la portabilitat del codi SQL.
- *Codificació de consultes*: en què veurem pràctiques que ens ajudaran a generar consultes SQL més eficients i llegibles.
- *Codificació de procediments i funcions*: en què pararem esment en aquelles pràctiques que ens permetran escriure el codi SQL en un servidor més eficient i gestionar errors de forma controlada.
- *Codificació de transaccions*: en què detallarem pràctiques per a assegurar un control correcte de les transaccions que codifiquem.

És important destacar que, dins de cada categoria, no es proporcionen totes les possibles bones pràctiques del mercat, sinó que es tracta d'un conjunt concret que des de la UOC hem considerat més rellevants.

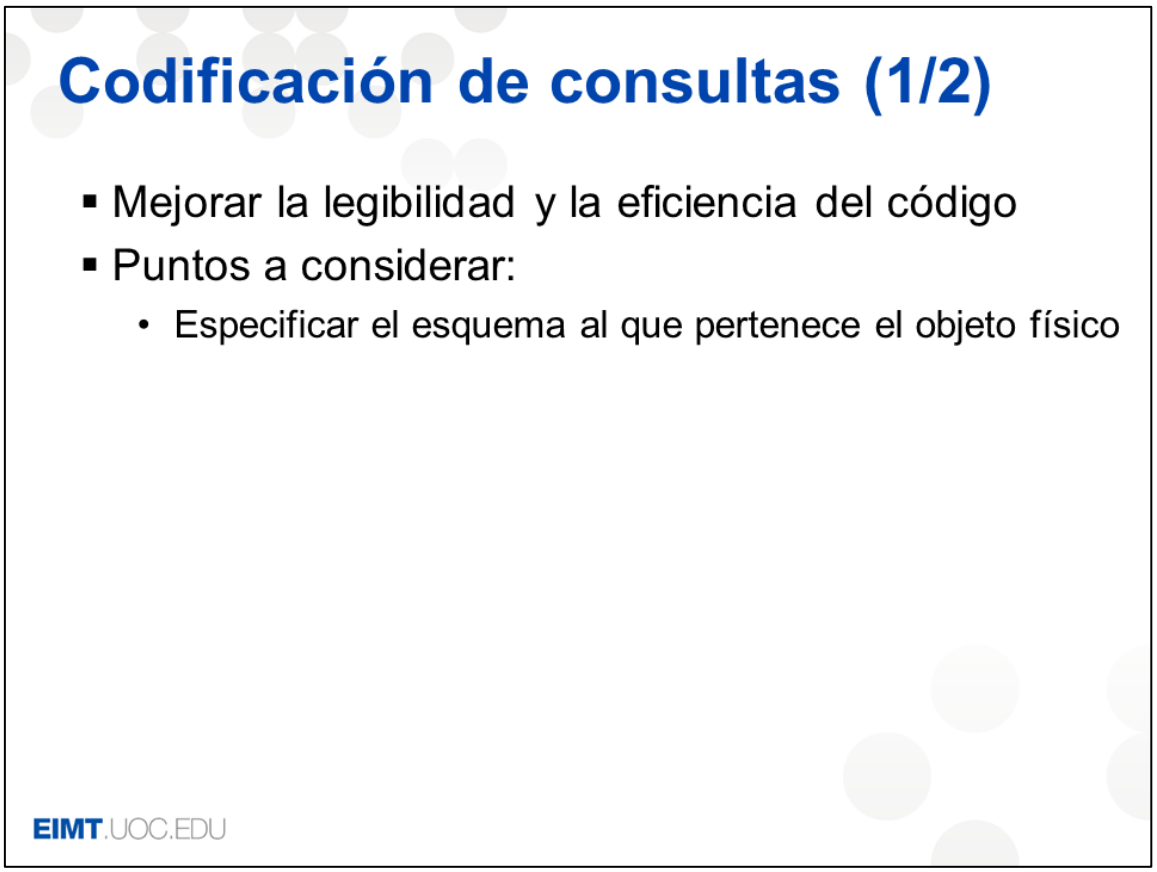
Aquest vídeo afrontarà la segona part, *Codificació de consultes*.



Aquest segon vídeo se centrarà a presentar una sèrie de bones pràctiques que ens permetran generar consultes SQL eficients, llegibles i portables, proposant en alguns casos exemples per a facilitar el seu enteniment. Al final del vídeo, us proporcionarem les referències bibliogràfiques utilitzades.

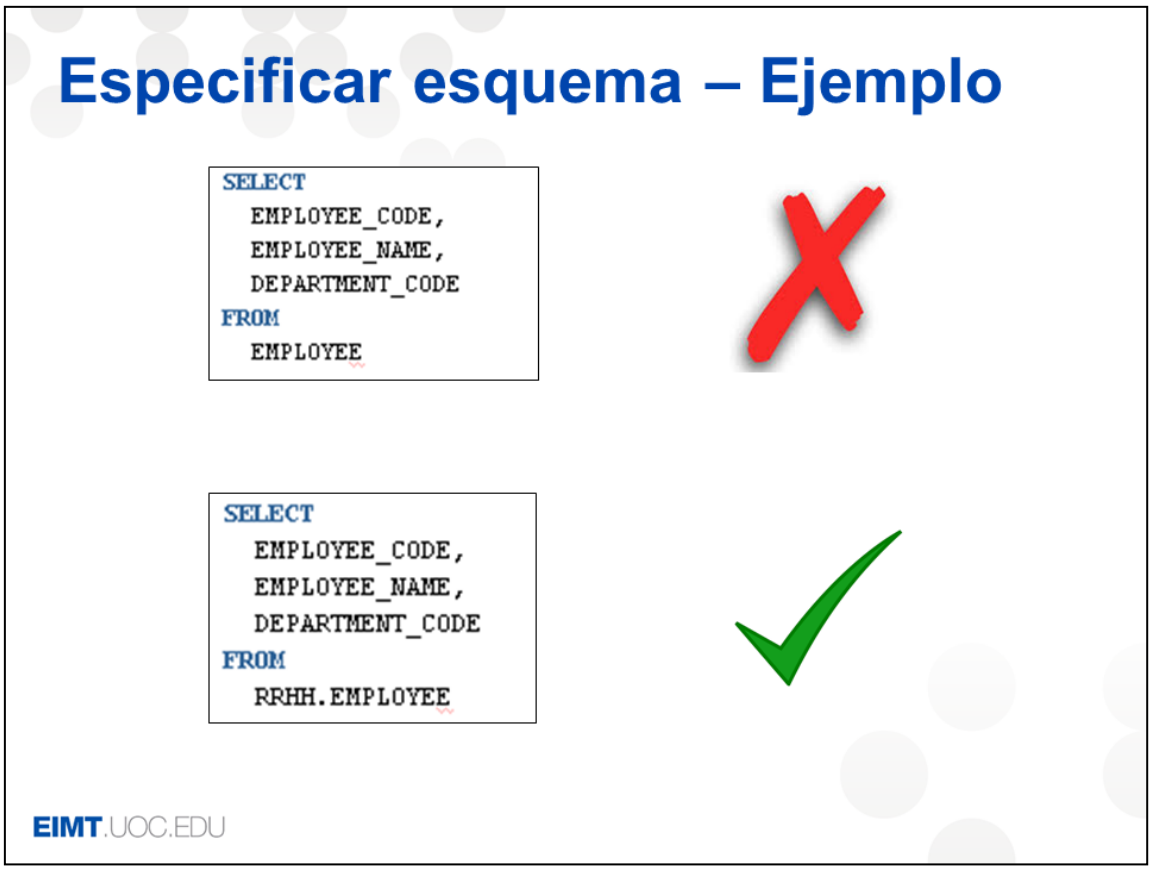


Vegem, doncs, la segona categoria de bones pràctiques: codificació de consultes.

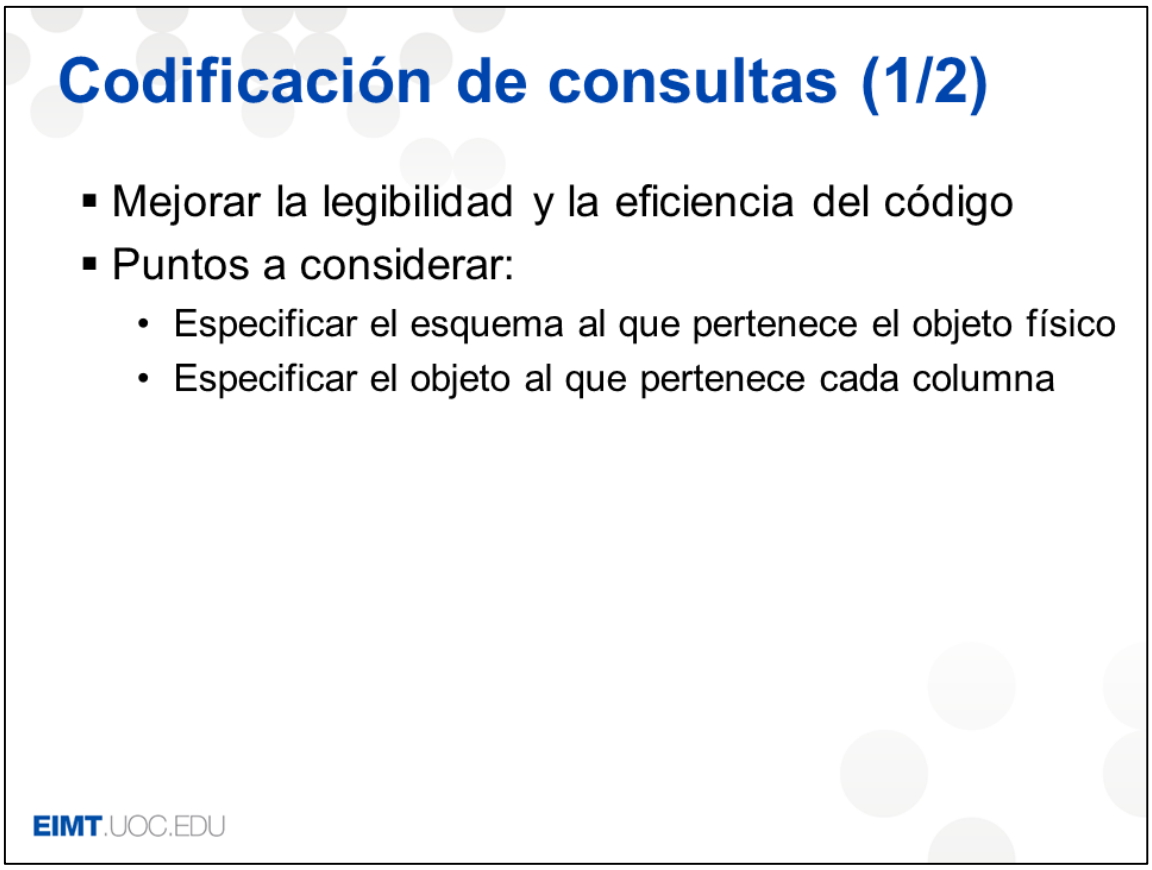


Aquesta secció presenta una sèrie de guies o pràctiques que ens permetran codificar consultes SQL més eficients alhora que comportaran una millor llegibilitat d'aquestes. Des d'aquest punt de vista, es recomanen els punts següents:

- *Especificar l'esquema a què pertany l'objecte físic*: quan realitzem una consulta SQL, generalment utilitzem taules, vistes o vistes materialitzades (entre d'altres), les quals són objectes que pertanyen a un esquema en concret. A l'hora de generar una consulta, es considera com a bona pràctica especificar l'esquema a què pertany aquest objecte. Vegem un exemple d'aquest punt.

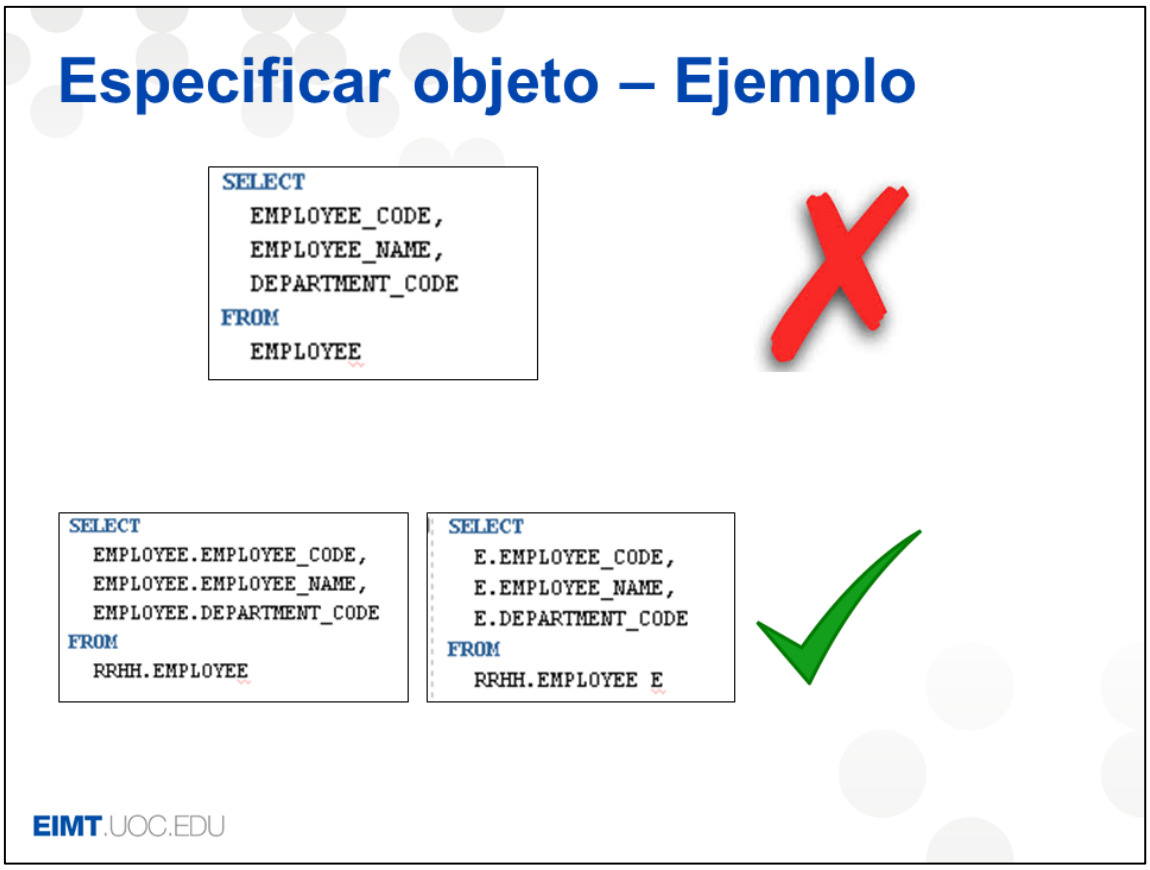


Suposem que tenim una taula *employee* que pertany a l'esquema *rrhh*. Una consulta que inclogui una crida a la taula *employee* s'hauria de realitzar mitjançant *rrhh.employee*. Això és així per a evitar problemes en cas que un usuari tingui accés a dues taules que hi ha amb el mateix nom però en esquemes diferents (per exemple, si hi ha una taula *employee* en l'esquema *taxes,* és a dir, *taxes.employee*).

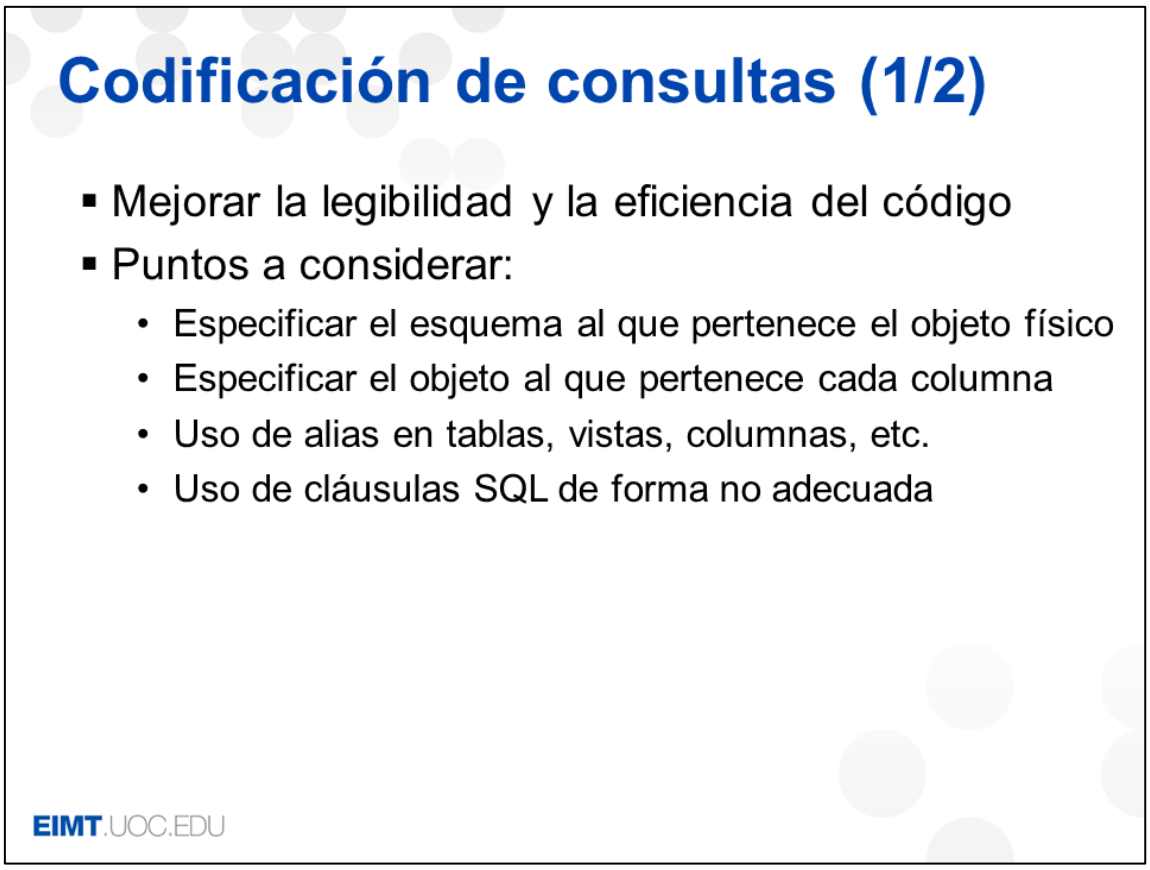


El punt següent és:

- *Especificar l'objecte a què pertany cada columna*: a l'hora d'especificar columnes en una consulta SQL (com a part de les clàusules `SELECT`, `WHERE`, `GROUP BY`, `HAVING` o `ORDER BY`), es recomana especificar a quin objecte pertany aquesta columna. Vegem un exemple d'aquest punt.



Utilitzant la mateixa taula *employee* vista anteriorment, la projecció de les columnes codi d'empleat, nom d'empleat i codi de departament s'haurien d'especificar amb el nom de la taula o bé amb un àlies (en cas que, per exemple, s'utilitzi la taula *employee* més d'una vegada en la mateixa consulta), tal com es mostra a les imatges.



Continuem amb els punts següents:

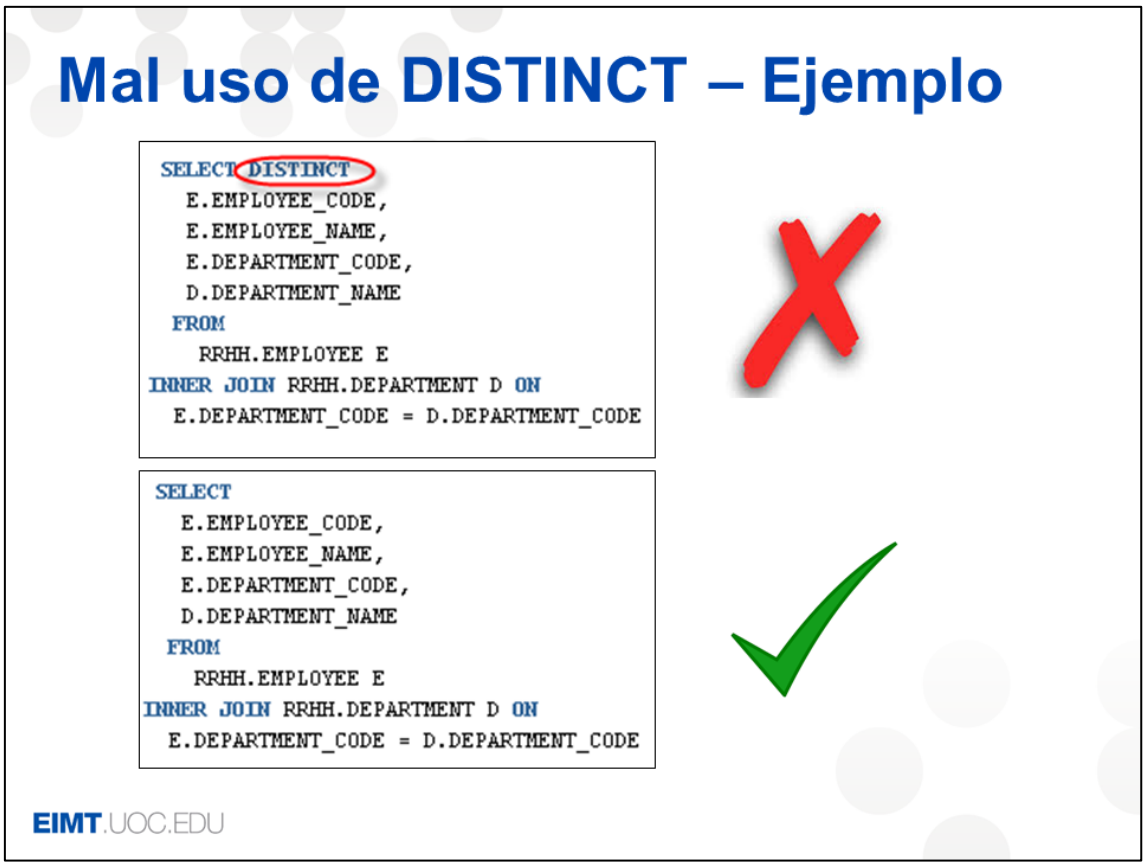
- *Ús d'àlies en taules, vistes, columnes*: en consultes SQL, podem especificar àlies tant en les taules, vistes o subconsultes, com en les columnes que es projecten. Es recomana l'ús d'aquests àlies per a facilitar la llegibilitat del codi i també per a evitar problemes de funcionament, en cas que una columna s'anomeni igual en més d'una taula, vista o subconsulta.

- *Ús de clàusules SQL de forma no adequada*: és molt important que les clàusules SQL siguin utilitzades per al propòsit que se'ls ha donat, ja que podrien causar els problemes següents:

  – Rendiment: la consulta realitza operacions de forma innecessària, exigint a l'SGBD un treball extra realitzant tasques que són innecessàries.

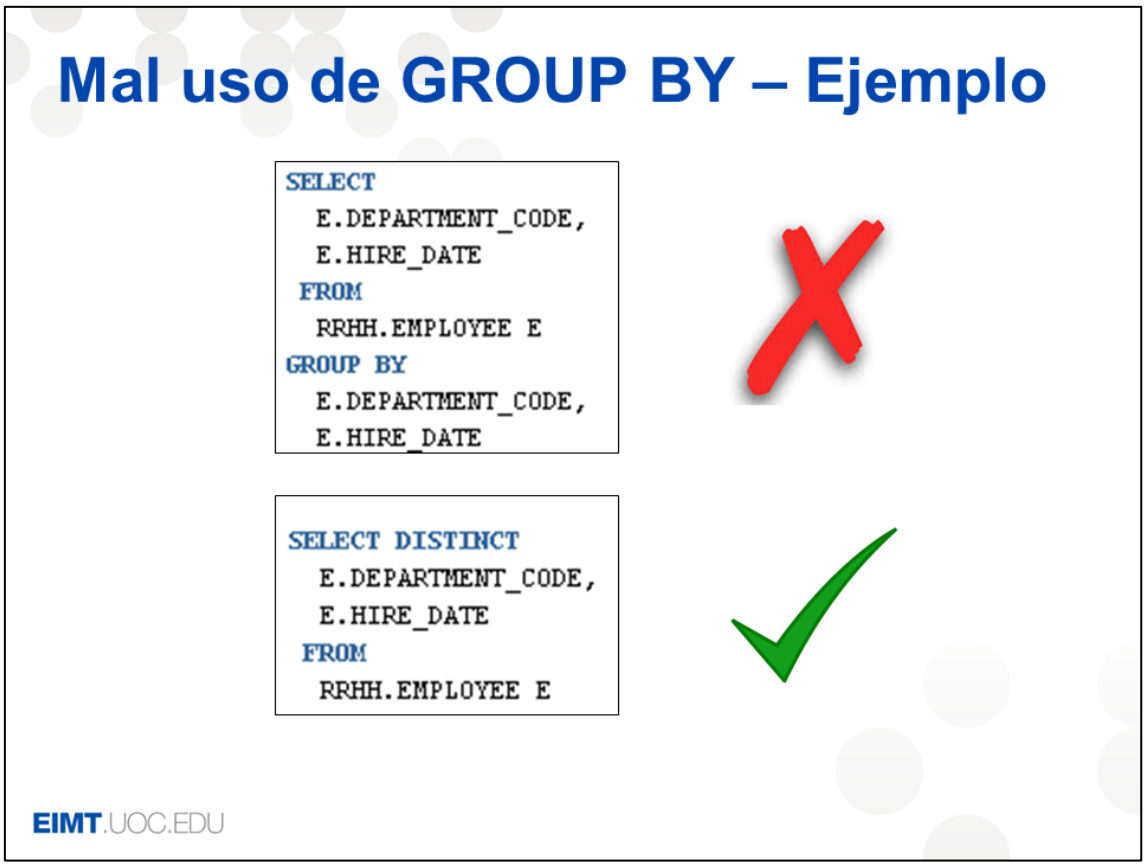
  – Ofuscació de codi: malgrat que és possible obtenir el mateix resultat utilitzant clàusules SQL diferents, un ús inadequat d'aquestes pot provocar problemes de llegibilitat i enteniment.

Vegem una sèrie d'exemples de l'ús de clàusules SQL de forma no adequada.

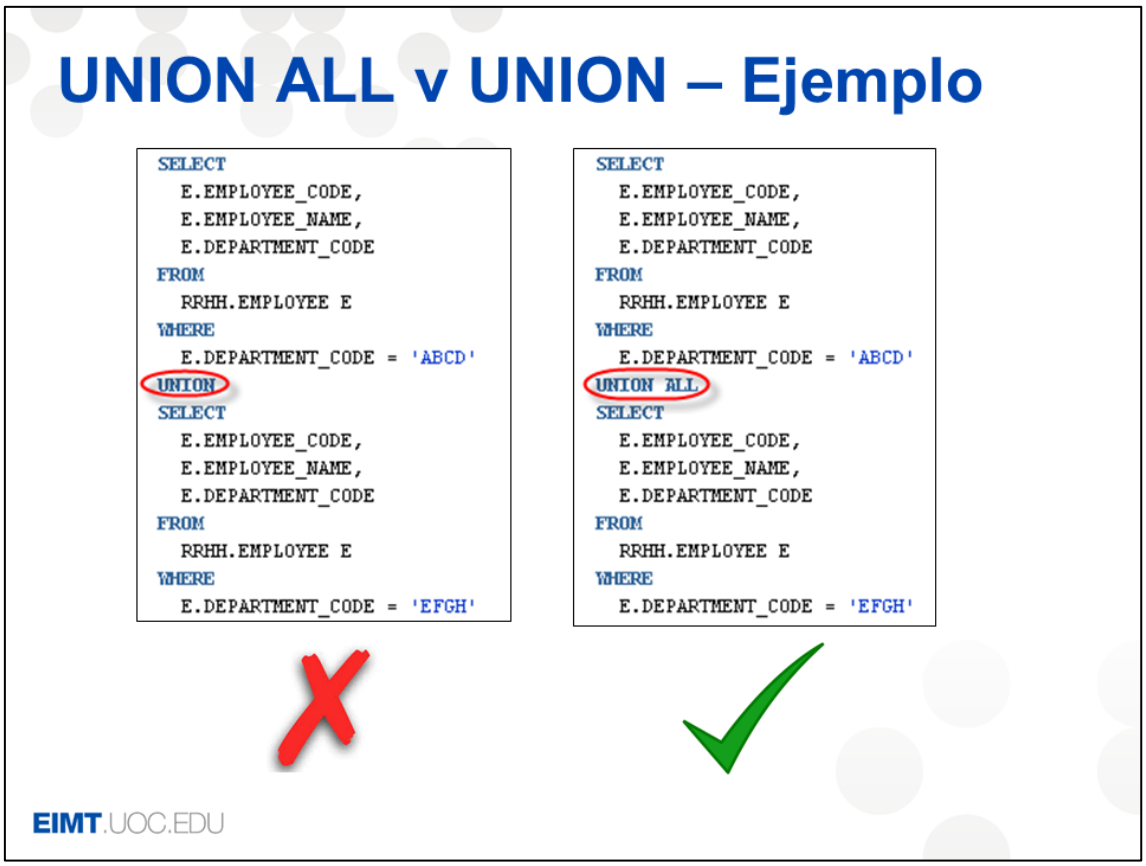


Suposant una consulta que obté informació d'empleats i departaments, és un error afegir la clàusula `DISTINCT` per a obtenir detalls dels diferents empleats quan la clau primària de l'empleat és part de la llista de columnes projectades de la consulta. En aquests casos, s'està exigint a l'SGBD que realitzi tasques per a distingir els diferents empleats de forma innecessària. Aquest tipus de problemes també se solen donar amb les clàusules `ORDER BY` i `UNION/UNION ALL` (`UNION` elimina duplicats, `UNION ALL` els manté).

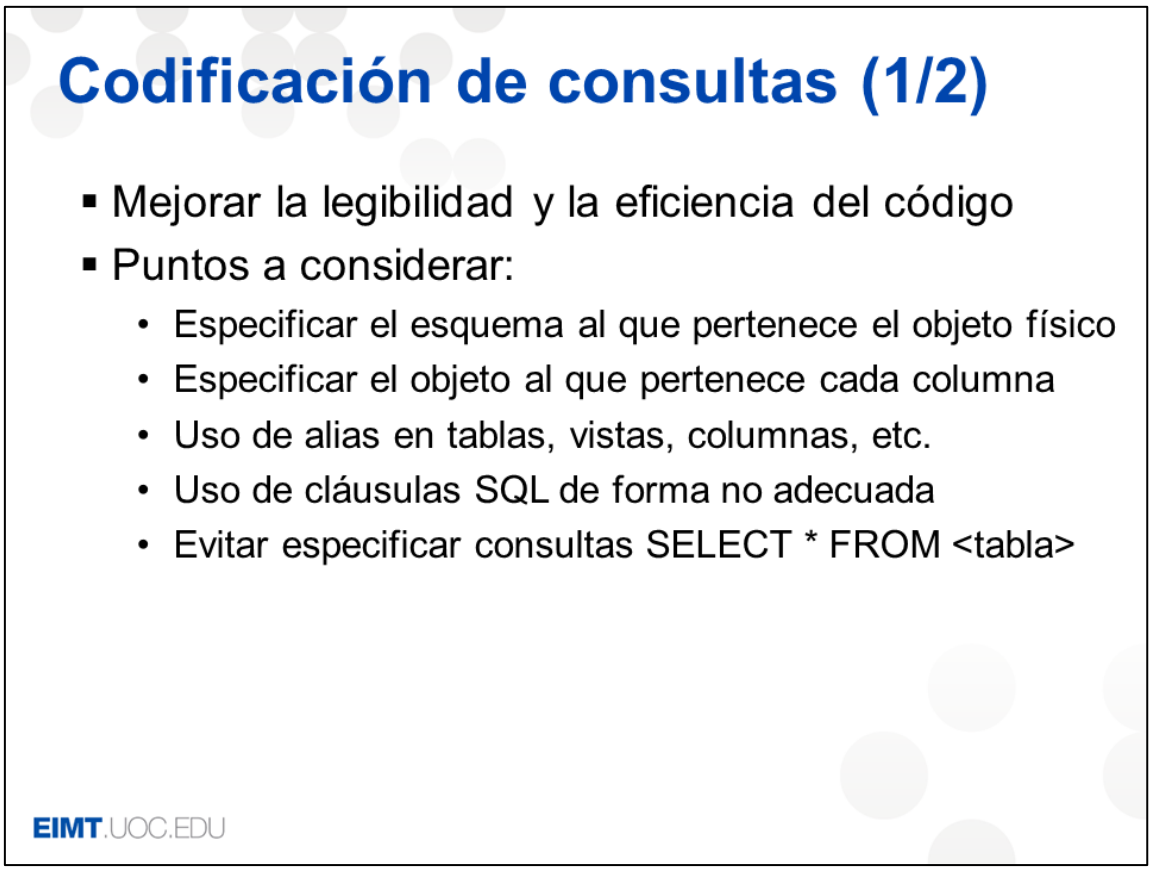
En resum: si l'ordenació de dades o eliminació de duplicats no són necessaris per a la construcció/significat de la consulta, aquestes clàusules han de ser eliminades de la consulta.



Suposem ara que volem obtenir els diferents codis del departament i la data de contractació dels nostres empleats per a estudiar els períodes de contractació de cada departament. La consulta mostrada a la part superior realitza aquesta tasca mitjançant un `GROUP BY`. Si bé ens proporcionarà els resultats esperats, el propòsit de la clàusula `GROUP BY` és el d'agregar informació i no el d'obtenir els diferents valors d'un conjunt de columnes. En aquests casos, el propòsit de la consulta és confús, per la qual cosa es recomana la utilització de la clàusula `DISTINCT` per a evitar malentesos.



Quant a l'ús d'`UNION ALL` i `UNION`: a l'hora de realitzar una operació `UNION`, és important entendre si el resultat esperat requereix que s'eliminin les files duplicades. En cas que no sigui necessari eliminar-les, es recomana l'ús d'`UNION ALL` en lloc d'`UNION`, la primera clàusula no elimina els duplicats sinó que simplement concatena els dos conjunts de dades, sense necessitat de combinar tots dos conjunts amb la finalitat d'eliminar les files duplicades. Això es pot veure en l'exemple següent, on s'obté la unió d'empleats de dos departaments diferents, en què mai no trobarem duplicats. L'ús d'`UNION` en la consulta de l'esquerra requereix que l'SGBD elimini els duplicats, mentre que l'`UNION ALL` de la consulta de la dreta simplement concatena els resultats d'ambdues subconsultes.

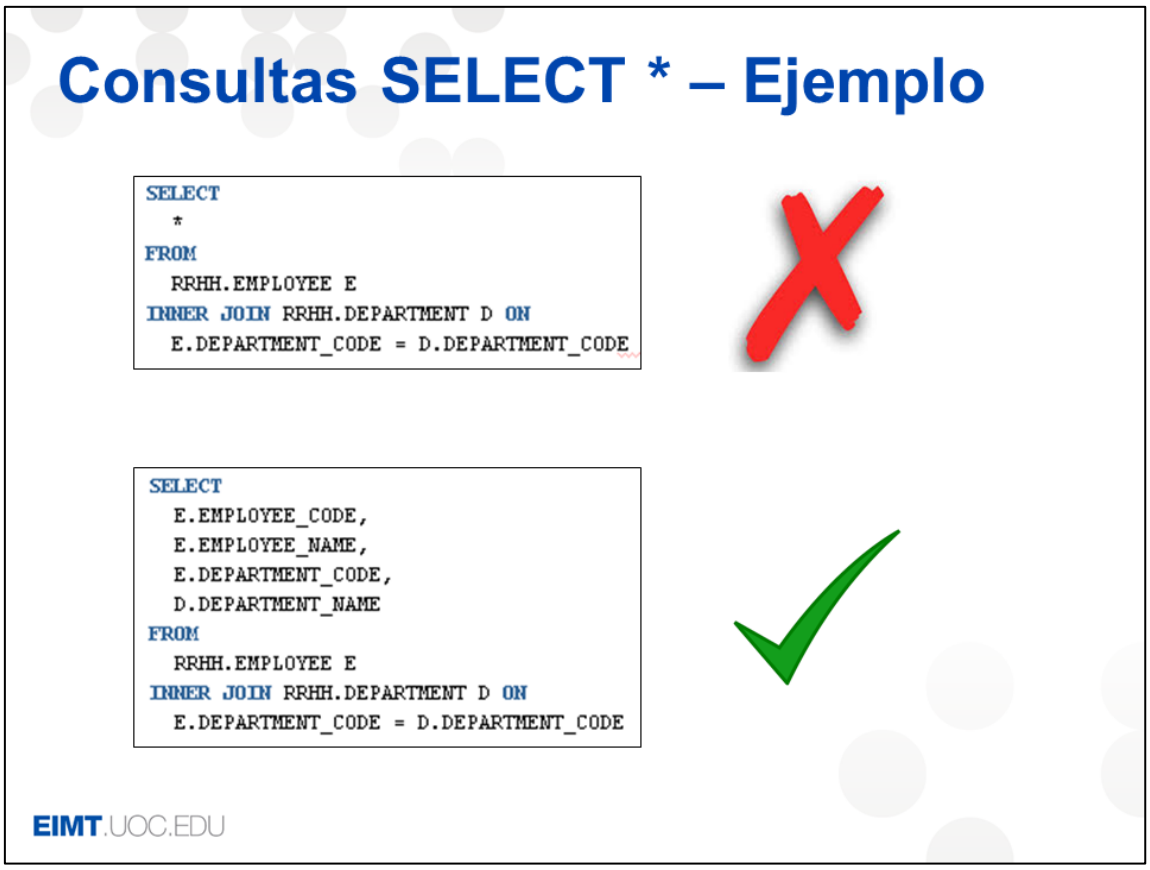


- *Evitar especificar consultes SELECT * FROM <taula>*: l'ús de consultes de l'estil `SELECT * FROM <taula>`, on s'obtenen tots els camps de les taules a què es fa referència, és bastant comuna en aplicacions de bases de dades. Cal evitar les consultes d'aquest tipus, ja que solen causar diversos problemes, entre aquests:

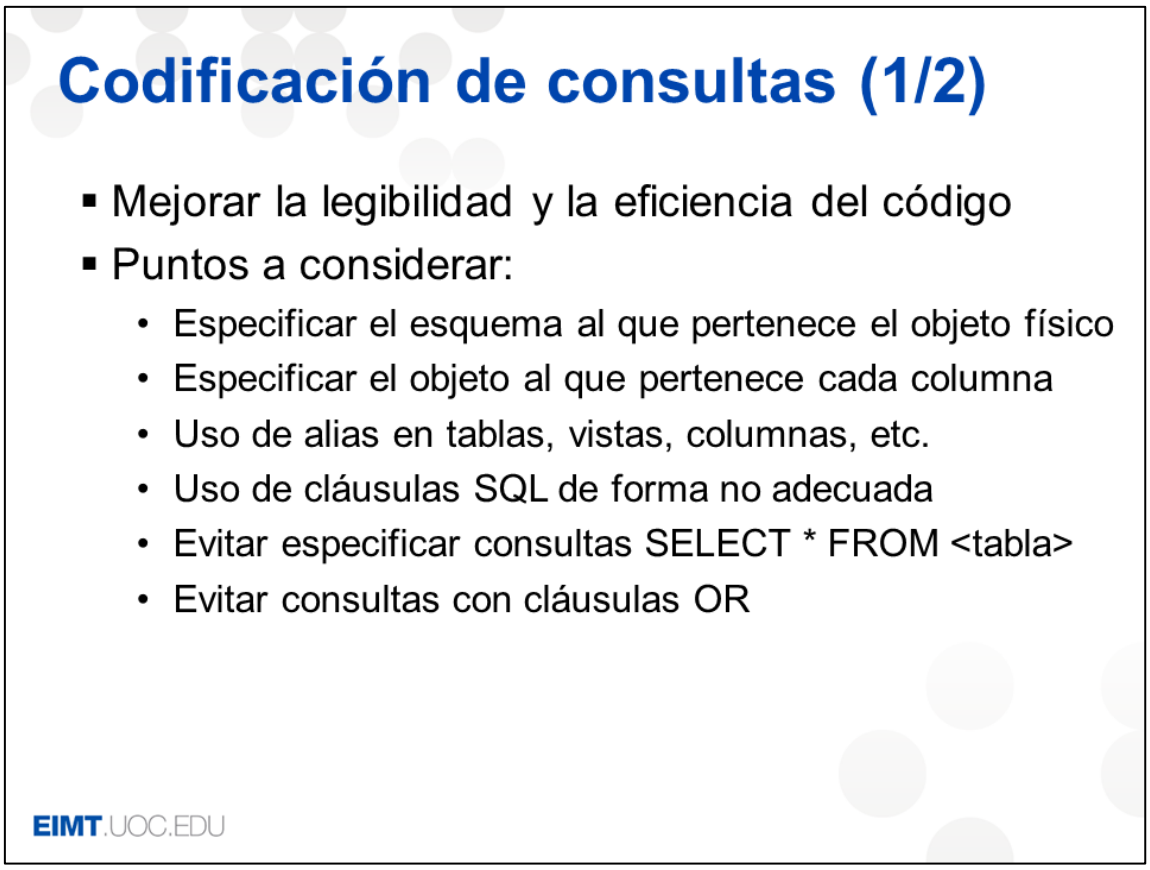
  – Generen més operacions d'E/S, perquè s'ha d'obtenir tots els valors de totes les columnes, malgrat que aquestes no s'utilitzin per part de l'aplicació. Això causa una petita deficiència en el rendiment de la consulta.

  – Poden causar problemes quan s'eliminen o s'afegeixen noves columnes en les taules, i l'aplicació que les utilitza no està dissenyada de forma correcta per a gestionar aquests casos.

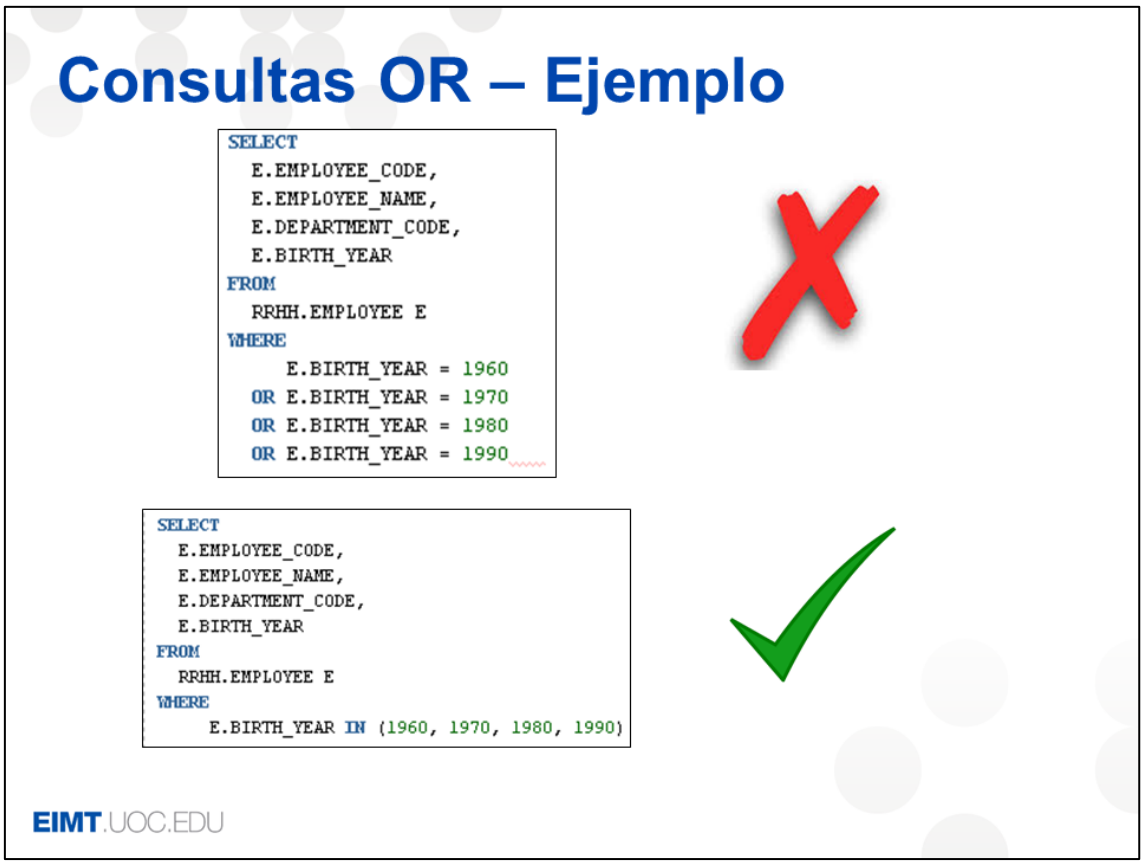
- Es recomana projectar solament les columnes que s'utilitzin. Vegem un exemple d'aquesta bona pràctica.



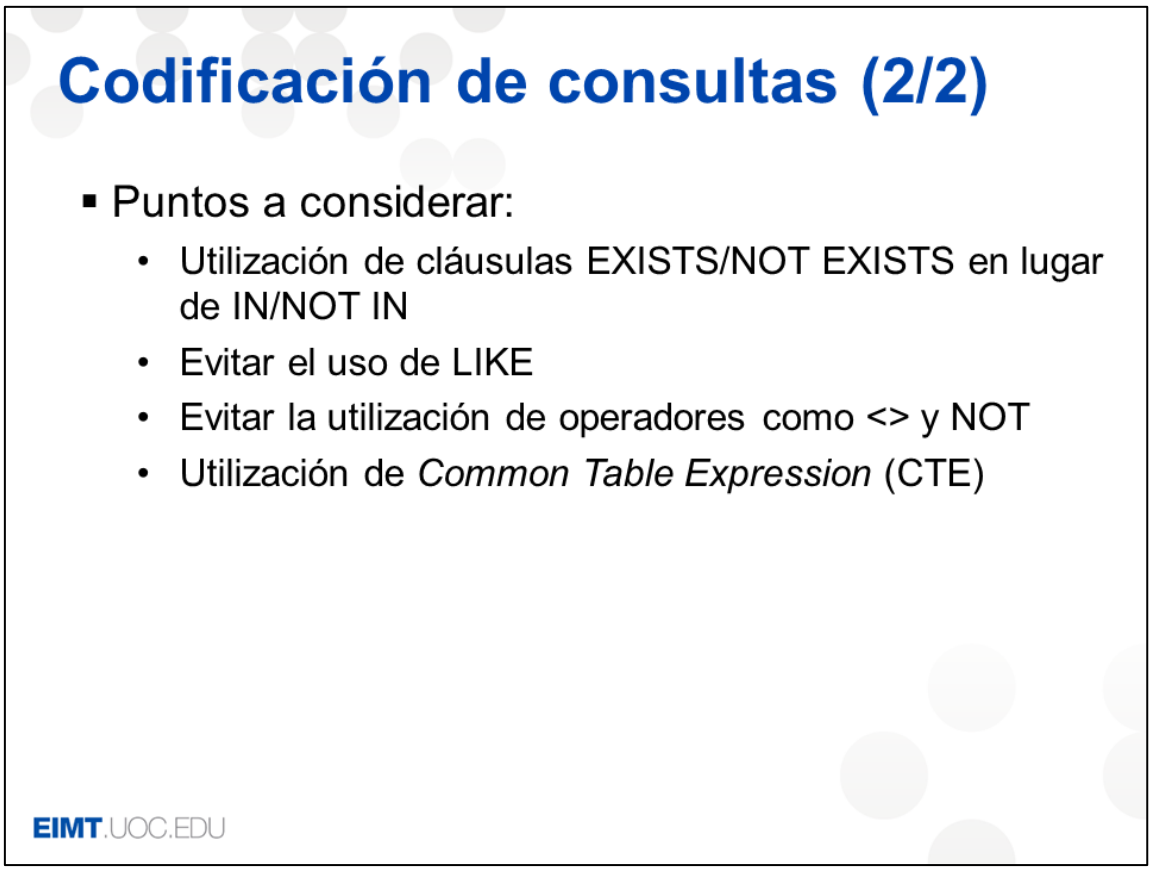
Suposem que volem obtenir el codi i el nom de l'empleat, i el codi i el nom del departament en què treballa. Per a obtenir aquesta informació, hem d'utilitzar les taules *employee* i *department*. A la imatge superior, la consulta projecta tots els camps d'ambdues taules (no solament els que es demanen). En aquests casos, es considera una mala pràctica l'ús de *, ja que ens retorna més informació de la que necessitem, a més que podria causar problemes en les aplicacions en cas que aquestes taules fossin modificades (per exemple, noves columnes o columnes eliminades). La consulta que s'hauria d'implementar es mostra en la segona imatge.



- *Evitar consultes amb clàusules OR*: a l'hora d'implementar consultes SQL, hi ha ocasions en què hem d'implementar condicions per a obtenir dades d'acord amb una sèrie de valors que pot prendre una columna en concret. Per a la implementació d'aquest tipus de consultes, es pot i se sol utilitzar l'operador `OR`. En casos com aquest, se sol recomanar l'ús d'`IN` en lloc d'`OR` , ja que no solament és més llegible, sinó que a més sol produir un millor rendiment, especialment quan la llista de valors és llarga. Fins i tot, es podria considerar l'opció d'`UNION ALL`, de manera que cada consulta que forma part de la `UNION ALL` sigui una consulta amb una condició d'igualtat sense `OR` per a un valor/conjunt de valors en concret. En qualsevol cas, sempre es recomana revisar el pla d'execució de cadascuna de les consultes per a verificar quina ofereix el millor pla d'accés.



Per exemple, si volem obtenir la llista d'empleats que han nascut els anys 1990, 1980, 1970 o 1960 respectivament, es podria crear la consulta utilitzant l'operador `OR` (com es mostra en la primera imatge). En aquesta consulta, si bé ens donaria els resultats esperats, veiem que hi ha una repetició de la mateixa condició. Això es podria simplificar utilitzant l'operador `IN` (segona imatge) que, generalment, l'SGBD implementa de manera més òptima que l'`OR`.



Continuem amb la llista de punts a considerar:

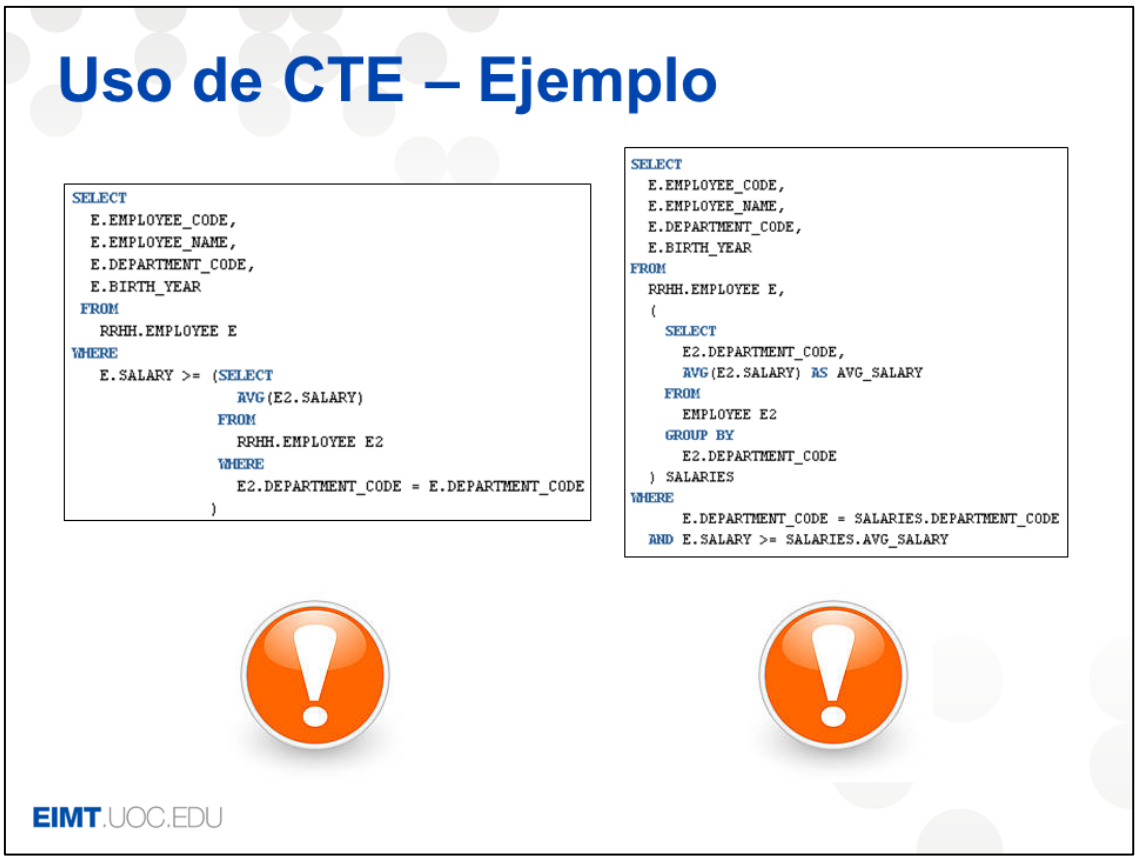
- *Utilització de clàusules EXISTS/NOT EXISTS en lloc d'IN/NOT IN*: generalment, l'ús d'`EXISTS` en lloc d'`IN` (i les seves variants negades) genera un millor rendiment, a més d'evitar la problemàtica del tractament de valors nuls, amb la qual podríem estar obtenint resultats incorrectes en cas que existeixin valors nuls en la columna utilitzada en la condició.

- *Evitar l'ús de LIKE*: l'operador `LIKE` ens permet verificar si les columnes de tipus caràcter compleixen una determinada característica mitjançant l'ús de patrons. L'ús de `LIKE` és costós, ja que requereix verificar dins de cada cadena de caràcters si es compleix la condició especificada i això requereix accés a les diferents posicions del valor comparat, per la qual cosa l'SGBD trigarà més temps a retornar els resultats. En la mesura que es pugui, l'ús d'aquest operador s'ha d'evitar per a no produir possibles problemes de rendiment.

- *Evitar la utilització d'operadors com <> i NOT*: l'ús d'operadors amb lògica negativa com ara <> i `NOT` en consultes solen produir algun que altre problema de rendiment, en comparació amb aquelles consultes en què s'utilitzen operadors amb lògica positiva (per exemple, =), sobretot en el cas que hi hagi índexs especificats en aquestes columnes, índexs que l'optimitzador de

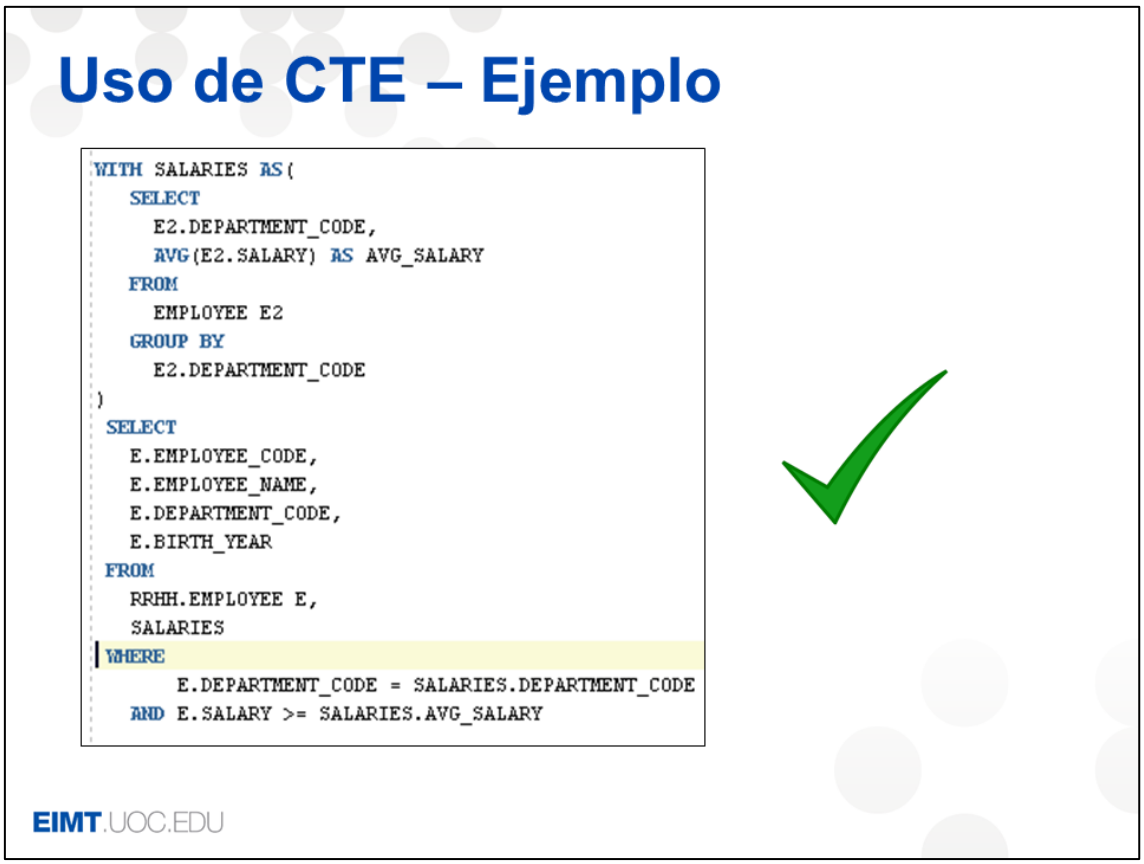
consultes no sol utilitzar quan es troba amb un operador negatiu. En la mesura que es pugui, haurem d'evitar aquest tipus d'operadors.

- *Utilització del common table expression (CTE) per a facilitar la llegibilitat del codi*: les taules derivades (subconsultes que apareixen en la clàusula `FROM`) són difícils de mantenir, a més de dificultar la lectura de la consulta. En aquests casos, es recomana l'ús de CTE per a facilitar la llegibilitat i el manteniment de la consulta. Vegem un exemple d'aquest punt.

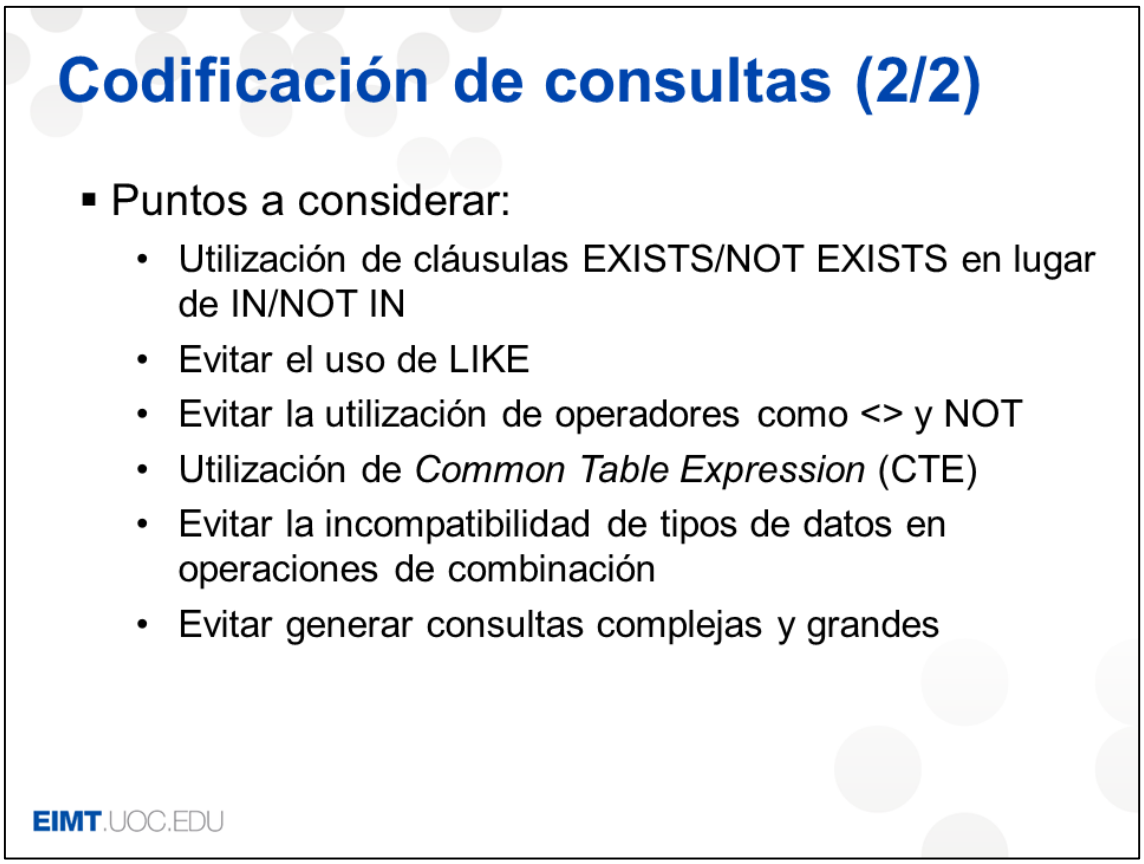


Suposem que volem implementar una consulta SQL que obté els empleats que tenen un salari igual o superior a la mitjana del salari del departament en què treballen. Aquesta consulta es podria implementar de diferents formes mitjançant l'ús de subconsultes. A la imatge de l'esquerra, veiem que per a calcular la mitjana per departament, s'ha utilitzat una subconsulta com a part de la clàusula `WHERE`. A la consulta de la dreta, s'ha utilitzat una taula derivada (una subconsulta com a part del `FROM`).

Ambdues consultes, si bé generen els resultats esperats, es podrien considerar poc llegibles o, fins i tot, podrien originar un manteniment més alt. A més, la primera d'aquestes no permet referenciar la informació de la mitjana d'empleats per departament en cas que aquesta informació necessiti referenciar-se en altres parts de la consulta principal, si bé la segona es podria referenciar perquè és part de la clàusula `FROM`.



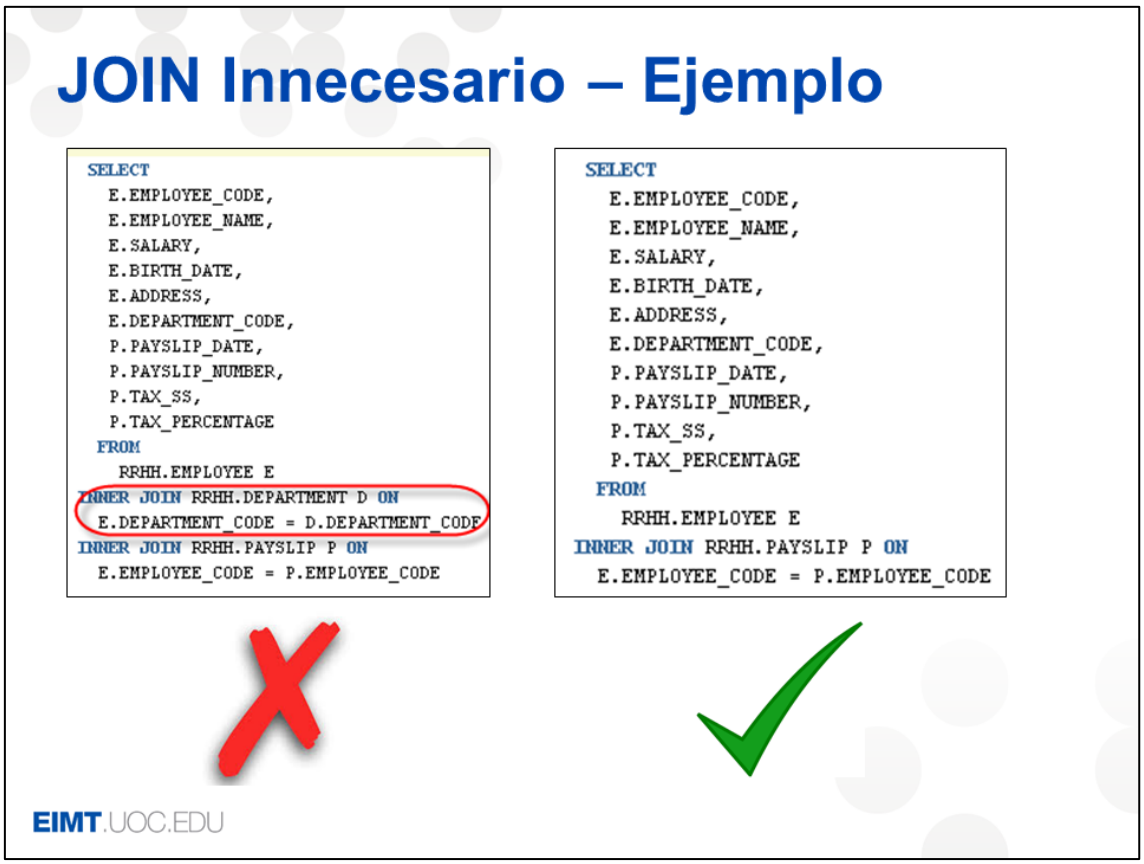
Aquestes dues consultes es podrien haver implementat mitjançant la tècnica de CTE amb la clàusula `WITH`, com es pot veure en aquesta transparència. Veiem que, en aquest últim cas, la consulta és més llegible, permetent diferenciar la subconsulta que calcula la mitjana del salari i la consulta principal, que obté el llistat d'empleats que compleix amb la condició requerida.



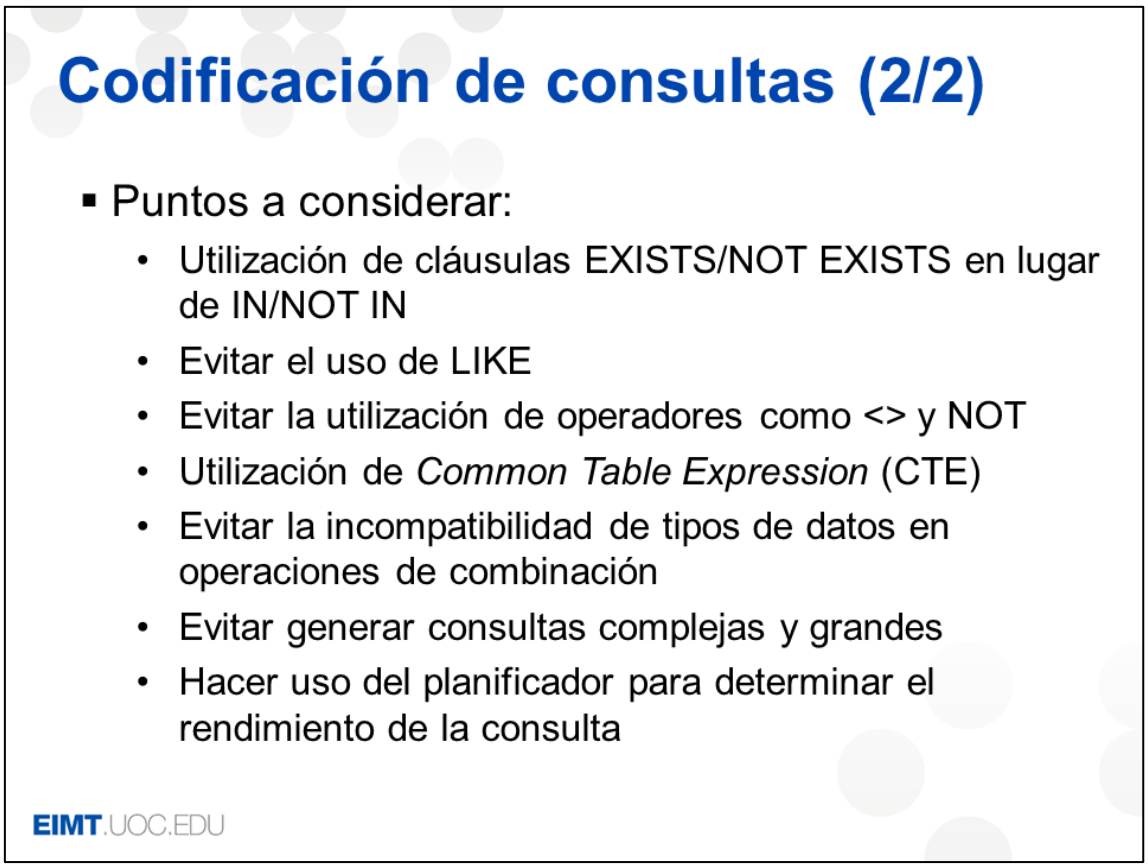
Continuem amb els punts següents:

- *Evitar la incompatibilitat de dades en operacions de combinació*: quan realitzem operacions de combinació entre taules, utilitzem un conjunt de columnes per a realitzar la comparació entre ambdues. Si els tipus de dades no són iguals, els SGBD solen transformar el tipus de dada d'una de les columnes perquè sigui compatible amb el tipus de dada de la columna que pertany a l'altra taula. Aquesta funcionalitat, malgrat que aparentment sigui útil, genera una degradació en el rendiment de les consultes a causa de la necessitat de transformar un tipus de dada a un altre.

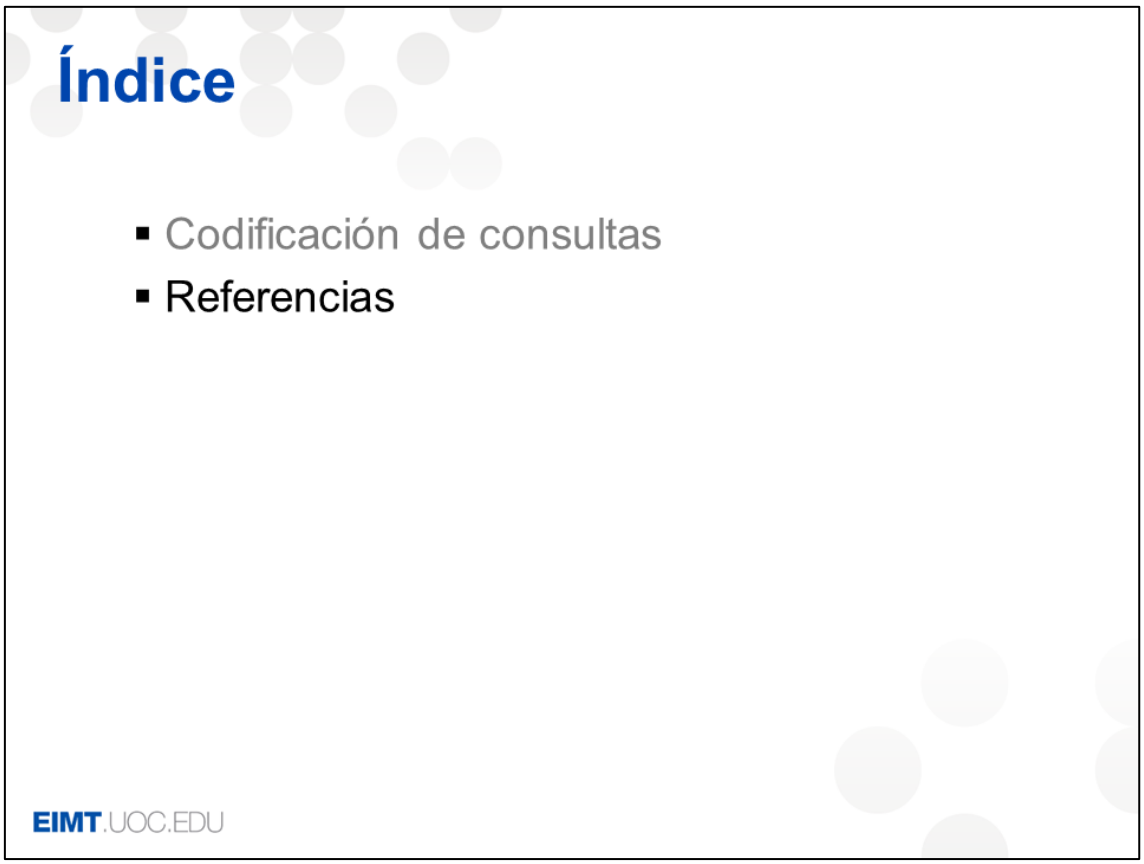
- *Evitar generar consultes complexes i grans*: consultes complexes i de gran grandària que utilitzen moltes taules i operacions de combinació poden requerir un temps considerable per a aconseguir que aquestes siguin el més òptimes possible. En aquests casos, és possible que l'optimitzador de consultes hagi d'utilitzar heurístiques per a generar el pla de la consulta, descartant així altres estratègies més òptimes, per la qual cosa és possible que el pla generat no sigui el més òptim. En aquests casos, cal sempre, en la mesura que es pugui, revisar si la consulta requereix utilitzar tots els objectes i operacions de combinació codificats. Vegem un exemple d'això últim.



Suposem que necessitem una consulta que ens proporcioni informació dels empleats (*employee*) i de les taxes aplicades en les nòmines (*payslip*), a més del nombre de nòmines i de la data emesa. La consulta que es presenta a l'esquerra ens proporciona aquesta informació, però veiem que a més realitza una operació de combinació amb la taula de departaments (*department*) de forma innecessària, ja que no es projecta cap informació procedent d'aquesta taula. La consulta de la dreta ens proporcionaria exactament la mateixa informació, a més d'evitar combinacions innecessàries.



- *Utilitzar el planificador per a determinar el rendiment de la consulta*: el planificador de consultes ens proporcionarà informació per a entendre el pla de la consulta i determinar el rendiment d'aquesta. Quan treballem amb consultes SQL, és molt important que utilitzem el planificador per a revisar el pla de la consulta i així decidir si la nostra consulta té un rendiment acceptable o no.



I fins aquí hem arribat amb aquest segon vídeo de la sèrie *Bones pràctiques en SQL*. Esperem que us hagi agradat la presentació i que aquesta us hagi servit de molta ajuda.

Ara us presentarem un conjunt d'enllaços i referències d'interès sobre aquest tema.

Que tingueu un bon dia!