**Ejercicios de Teoría de Conjuntos**

All = {1,2,3,4,5,6,7,8,9,10}

Even = {2,4,6,8,10}

Odd = {1,3,5,7,9}

Realice las siguientes operaciones:

1. Even U Odd

Se toman todos los elementos que estén en Even o en Odd, sin repetir.

Even∪Odd={1,2,3,4,5,6,7,8,9,10}

1. Even \*\*∩\*\* Odd

Se toman solo los elementos que estén en ambos conjuntos a la vez.

Even∩Odd=∅

1. All – Odd

Todos los elementos que están en All, pero no en Odd

Resultado = {2,4,6,8,10}

1. C(Even)

todos los elementos de **All** que **no** están en **Even**

C(Even)={1,3,5,7,9}

1. C(Odd-All)

Calcular la diferencia (Odd – All) Odd−All=∅

Luego obtener el complemento de ese resultado respecto a All. La diferencia **Odd − All** significa: “elementos que están en Odd pero no en All”.

C(Odd−All)={1,2,3,4,5,6,7,8,9,10}

Ejercicios de JOINs

1. Investigue y busque documentación sobre el SELECT que explique cómo utilizar las funcionalidades de ORDER BY, LIMIT, GROUP BY y tres tipos de JOIN, y cree una tabla (informativa, no de SQL) a partir de sus descubrimientos.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Funcionalidad** | **¿Para qué sirve?** | **Sintaxis base (genérica)** | **Notas clave** | **Diferencias por motor** |
| **ORDER BY** | Ordena las filas del resultado. | `SELECT ... FROM ... ORDER BY expr1 [ASC | Acomoda las filas | Sin ORDER BY, **el orden no está garantizado**. En PostgreSQL puedes controlar nulos: NULLS FIRST/LAST. |
| **LIMIT / OFFSET** | Restringe la cantidad de filas devueltas (y salta las primeras OFFSET). | ... ORDER BY ... LIMIT n OFFSET m | Para resultados **deterministas**, úsalo junto con ORDER BY. | **PostgreSQL:** documenta el riesgo de subconjuntos “impredecibles” sin ORDER BY. **MySQL:** el optimizador intenta elegir índices ordenados con LIMIT + ORDER/GROUP BY. **SQLite:** comportamiento estándar en SELECT. (PostgreSQL, MySQL Developer Zone, SQLite) |
| **GROUP BY** | Agrupa filas para aplicar agregaciones (SUM, COUNT, AVG…). | SELECT col, AGG(...) FROM t GROUP BY col | Solo columnas agrupadas o agregadas deberían ir en el SELECT. Usa HAVING para filtrar **después** de agrupar. | **MySQL:** permite seleccionar columnas no listadas en GROUP BY (extensión propia); también soporta WITH ROLLUP. **SQLite:** uso estándar de GROUP BY con agregados. (MySQL Developer Zone, SQLite Tutorial) |
| **INNER JOIN** | Devuelve solo las filas que **coinciden en ambas** tablas (intersección). | ... FROM A INNER JOIN B ON A.key = B.key | INNER suele ser implícito. | [PostgreSQL: INNER es el valor por defecto. SQLite/MySQL: soporte estándar. (PostgreSQL)](https://www.postgresql.org/docs/current/queries-table-expressions.html?utm_source=chatgpt.com) |
| **LEFT (OUTER) JOIN** | Devuelve **todas** las filas de la tabla izquierda y las coincidentes de la derecha; si no hay match, columnas derechas a **NULL**. | ... FROM A LEFT JOIN B ON A.key = B.key | Útil para detectar “sin pareja” con WHERE B.key IS NULL. | [PostgreSQL: comportamiento canónico. SQLite/MySQL: soporte estándar. (PostgreSQL)](https://www.postgresql.org/docs/current/tutorial-join.html?utm_source=chatgpt.com) |
| **FULL (OUTER) JOIN** | Devuelve **todas** las filas de **ambas** tablas, rellenando con **NULL** cuando no hay coincidencia. | ... FROM A FULL JOIN B ON A.key = B.key | Equivale a LEFT ∪ RIGHT. | **PostgreSQL:** soportado. **SQLite:** soportado desde **3.39.0 (2022)**. **MySQL:** **no** tiene FULL OUTER JOIN nativo; se emula con LEFT JOIN **UNION** RIGHT JOIN. (PostgreSQL, SQLite, SQLite, MySQL Developer Zone, Stack Overflow) |

Fuentes clave: documentación oficial de **PostgreSQL** para SELECT/ORDER BY/LIMIT y JOINs; **SQLite** lang\_select (y notas de versión 3.39.0); **MySQL** manual de SELECT/JOIN, optimización de LIMIT, y confirmación común de que **no** soporta FULL OUTER JOIN