Knowing the **Order of Processing** off by heart will prevent you from having to memorize which clauses can access which aliases and understand how the queries narrow and search data.

**Order Of Processing:**

**1. FROM -> For:**

**choose tables to gather base data**

**2. JOIN -> Jack:**

**same as above**

**3. WHERE -> Will:**

**filters the base data, prior to any grouping**

**4. GROUP BY -> Grow:**

**aggregates/groups the base data**

**5. HAVING -> Having:**

**filters the aggregated data**

**6. SELECT -> Selected:**

**returns the final data**

**7. DISTINCT -> Dinner:**

**provides non-duplicated final data**

**8. ORDER BY -> Over:**

**sorts final data**

**9. FETCH -> Fun:**

**limits the returned data to row count**

**Generalized Test Tips:**

* Be skeptical of answer choices which are overly inclusive or exclusive (i.e. choices with words including superlatives like ALWAY, MUST, NEVER, etc.); Oracle loves having little loopholes in their rules to throw you off; it seems like those options are often red herrings
* If you have become a query “master” prior to the test, skip all query questions your first time through and focus on the knowledge questions; the queries will require less mental energy because you’re often just trying to parse long queries with minute differences; queries tedious, but not mentally taxing; the knowledge questions require a little more mental energy to really parse the answers they’re looking for; spend that mental currency up front and then go back to the queries
* Pay close attention to questions which ask what the outcome will be; REMEMBER: just because the outcome isn’t what you want, it might still run
* Use the tools that the exam software provides you! Right-click to strikethrough options as soon as you know it is completely incorrect; this can quickly eliminate questions and prevent wasting mental bandwidth

**ERD vs. DBs:**

* In M:M relationship, you have to reduce to two 1:M relationships and then have an associative table to connect them
* ERDs don’t require anything that could be considered DB focused lingo (table, keys, etc.); ERDs are focused on **relations, entities, and attributes;** it’s all theoretical

**Tables:**

* **TABLE INSERTS:**
  + Insert All evals all when statements; Insert First evals until when statement is true
  + Multitable inserts can be performed on relational tables NOT on views, materialized views, or remote tables; Can only use subqueries and each computed row can be inserted into multiple tables (it is a multitable insert after all!)
  + Rows inserted into a TABLE via a VIEW are retained when the VIEW is dropped;
* **TABLE DROPS**:
  + Views, Synonyms = **INVALIDATED**
  + Indexes, Constraints = **DROPPED**
  + The table **“may”** be moved to recycle bin; the recycle bin can be **DISABLED**, however, in which case the table would **NOT** move to the recycle bin automatically after a drop
  + During flashback, all table constraints **except referential integrity constraints that reference other tables** are retrieved
* Can't use hyphens or Oracle Keywords in column alias unless within double quotes
* Adding a new not null column in a table with data requires a default value; You can change the default value of a column; You can increase the width of a numeric column
* CANNOT rollback column drop, it's **permanent**!
* The FROM in a DELETE statement is optional; the DELETE command also doesn't require a WHERE clause
* **EXTERNAL TABLES:**
  + External tables are tables which are accessible from the RDBMS, but the actual data is not stored there. All metadata about the table is stored within the RDBMS.
  + **You cannot INSERT data into an external table!**
* **TEMPORARY TABLES:**
  + **Global Temporary Table (GTT):**
    - The structure is visible to the ALL current sessions connected to DB, but the DATA within is private to each session.
    - GTTs are placed in storage
    - When a session ends, a GTT table is truncated of its data, but the structure remains
    - For GTTs you can use the "ON COMMIT" clause to have it "PRESERVE ROWS" or "DELETE ROWS". When a **COMMIT** happens, the option will execute. If ON COMMIT is set to PRESERVE, the rows won't be deleted until the current session ends. This is referred to as **SESSION-SPECIFIC**, the other is called **TRANSACTION-SPECIFIC**.
  + **Private Temporary Tables (PTT):**
    - The structure and data is visible ONLY to the session which created it
    - PTTs are placed in memory
    - When a session ends, a PTT is completely gone (data and table definition)
    - In PTTs the commit options are "DROP DEFINITION" or "PRESERVE DEFINITION"
* In READ-ONLY mode, DDL statements are allowed, DML are not. Think of truncate as a delete rows command; Setting columns to unused is not allowed, BUT dropping already unused columns is allowed

**Joins:**

* Natural joins will join any columns that have the same name and the same data type OR a type which can be implicitly converted (i.e. NUMBER to VARCHAR2)
* Equijoin & Nonequijoin are both types of **INNER JOINS**
* Equijoin: Combining one or more of the same columns from different tables; can have other conditions in the where clause that don't involve the combination of columns
* Nonequijoin: combination of two tables along unequal terms (no shared columns); the where clause uses a non-equality (=) operator to join the tables such as >, <, along with all, any, in, between, etc.
* In a self-join, there must be a condition on which the self-join is performed (i.e. an ON clause)

**Subqueries:**

* In correlated subquery the nested query executes second because it needs the data from the outer query (remember it's correlated, so it is related to the outer query)
* Multiple row subqueries return ONE OR MORE rows; can have GROUP BY and HAVING clauses
* EXISTSis more performant than IN/NOT IN, specifically on large data sets. Often used when dealing with subqueries (particularly correlated subqueries)
* EXISTS finishes running when it finds a result and returns TRUE **OR** once it finished on the subquery data and then returns FALSE;
* If a NATURAL JOIN were being used which would be matching more columns than desired, one could use a JOIN with USING and restrict the number of columns being matched
* The USING clause takes a single column name which is the same in each table (name and definition); if the above can't be met, the ON clause can be used when column names or definitions between the tables are different AND it can use conditional arguments; the ON clause can perform some implicit data type conversions

**SET Operators:**

* “Compound Queries” are queries which use SET operators
* UNION: An 'OR' operation that eliminates duplicates between two tables; sorted
* UNIONALL: An 'OR' operation which does NOT eliminate duplicates; unsorted
* INTERSECT: An 'AND' operation
* MINUS: Feels similar to left outer join; first table (A) has any rows shared with B removed, all remaining rows from A are returned; changing the table order CAN change the results
* **UNION does not ignore null values during duplicate checking**
* A UNION following a UNION ALL will remove any duplicates from the initial pairing
* The data types of each column have to be of the same **group** (i.e. CHAR and VARCHAR); no implicit conversions across different data type groups!
* Order By clause of Compound Query references **first** SELECT clause
* If first SELECT clause has a column alias using single-quotes (‘ex\_col\_alias’), you **CANNOT** reference that column alias in ORDER BY via its value; you must use its positional number
* If the column alias uses double quotes (e.g. “ex\_col\_alias”), you reference it without the quotes in the ORDER BY clause, just like normal

**Views:**

* Expressions are **NOT** valid as column names in views (i.e. COUNT(\*)); **must use alias**
* Views are considered objects, so they have an object number
* Views are not allocated space, so they don’t have a segment number
* Rows inserted into a TABLE via a VIEW are retained when the VIEW is dropped;
* When a table associated with a view is dropped, the view is invalidated, but retained

**Sequences:**

* If a user is trying to INSERT data into a table which contains a DEFAULT value of a sequence, the user must have been granted access to that sequence.
* If a sequence is dropped in the above scenario, an insertion attempt with **no provided value** will fail and return "sequence does not exist"
* DROP command is used to delete sequences
* Sequences can be used for more than inserts
* Once nextval is called, that is the value
* Sequences only produce duplicates if they CYCLE over their MAXVAL
* They can (and probably will) have gaps because they can be used in different tables simultaneously
* Unallocated cached values are lost if the DB INSTANCE shuts down

**Indexes:**

* Indexes are invoked via FROM and WHERE; **NOT** in SELECT, GROUP BY, or ORDER BY
* If you specifically indicate an index for use with a newly created pk, it uses that index and trashes the automatically made one; even if you use the CREATE INDEX command as a subquery of the USING INDEX command

**Merge:**

* MERGE must have ON clause; WHEN MATCHED needs UPDATE
* During MERGE INTO you cannot UPDATE the column which is used as the condition in the ON clause; **this would obviously cause problems!**
* MERGE can use SUBQUERIES and VIEWS as **source rows**;
* MERGE can update, insert, and delete rows conditionally in multiple tables via a subquery

**With:**

* Think of this as a temporary view or table which is saved into memory for use within the query that you're making. It not only cleans up the appearance of more complicated queries, it also ends up running the larger query much faster because the subquery is saved in memory.
* The WITH clause can only be used with the SELECT clause **NOT** UPDATE, FROM
* It can have more than 1 query
* The query name in WITH clause is visible to blocks within and without said the same clause

**Constraints:**

* Primary Key columns are equivalent of NOT NULL and UNIQUE
* Foreign Key Columns can have NULLs
* Unique Key Columns can have NULLs
* Primary Key columns can be dropped; if the question doesn’t reference constraints, then the column can be dropped
* Child rows don't have to be deleted concurrent with a parent row; their FK value can be CASCADE SET NULL instead
* Column level constraints (NN) must use the MODIFY keyword followed by the column name in an ALTER TABLE statement; the order is otherwise unchanged
* C in the constraints table represents a check constraint (NN counts as a check)
* DESC can be used in both SQL\*Plus AND SQL Developer; displays only NOT NULL constraints, column names, and data types;
* During flashback, all constraints **except referential integrity constraints that reference other tables** are retrieved

**NULLS:**

* No group functions pay attention to NULLs except COUNT, but only if you use the wildcard (e.g. COUNT(\*))
* All group functions can be made to pay attention to NULLs if you use an NVL or NVL2 first
* Foreign Key Columns **allow NULLS**; Unique Key Columns **allow NULLS**
* NVL/NVL2 requires the same data types, generally; NVL2 will try an implicit conversion from the 3rd arg type to the 2nd arg type, **NOT THE OTHER WAY**
* NVL arguments aren't optional
* Adding a new not null column in a table which already contains data **requires** a DEFAULT value
* If you see any queries with WHERE clauses containing “... = NULL” or anything other than “IS NULL” or “IS NOT NULL”, those are wrong
* Values can be set to NULL with “= NULL” (i.e. in an Update statement)

**Single-Row Functions:**

* Single-row functions return single result row per row in the table
* Single-row functions can be nested to any level
* Single-row functions are things like CEIL, FLOOR, TRUNC, TO\_CHAR, etc.
* The data type returned can be different than input (e.g. TO\_DATE(‘01-Jan-2000’));
* They can accept more than one argument (e.g. CONCAT(‘10293’, ‘2nd Arg’));
* They can accept column names, variables, literals, or expressions (e.g. TO\_CHAR(column\_name); TO\_CHAR(‘Literal’); TO\_CHAR(&col\_name\_var); TO\_CHAR(5+5));
* ROUND and TRUNC default to 0th index (i.e. decimal point); a negative index means ones, tens, or hundreds as you go more negative; Rounding rule: up is >= 5; TRUNC always rounds down **AND** TRUNC works with dates
* TRIM truncates leading or trailing or both characters which are specified; if left blank it removes leading and trailing whitespace; use LEADING, TRAILING keywords to be specific; you can use the BOTH keyword, but leaving it blank is equivalent

**Group Functions:**

* No group functions include NULLS unless you NVL the NULLS
* Group functions can only be nested to **TWO LEVELS**
* You can have multiple group functions in a select statement; They can use columns or expressions; they don’t require a group by if it’s just group functions...**unless there’s nested group function, then order by is required**
* **Nested group functions REQUIRE A GROUP BY CLAUSE!!!**
* HAVING clause is very similar to the WHERE clause, but it uses aggregate functions to filter the data that is aggregated in the GROUP BY clause.

**Conversions And Data Type Arithmetic:**

* Date +/- Num = Date; Date-Date = Num; **Date+Date = NOT ALLOWED!**
* Dates are implicitly convertible from a string if the month in the string is in MON or MONTH format, strict numbers will not work (e.g. ‘01-Jun-2000’, ‘01-June-2000’; **NOT** ‘01-06-2000’)
* **Date type can be implicitly converted to a number type**

**Intervals:**

* Intervals allow you to create chunks of time which you use as arithmetic operators on DATES and TIMESTAMPS. You can add/subtract INTERVALS to/from Dates and Timestamps. You can multiply/divide INTERVALS by numbers.
* Two types: **Year To Month** and **Day To Second**
* Year To Month can provide an interval starting with years and going down to months ('1-2' = 1 yr, 2 months). Default precision for both numbers is 2 digits. Year can be 0-9, and Months can be 0-11. Can simply be years or months

**Syntax:**

**Interval ['interval\_str\_here'] Year[(0-9)] To Month[(0-11)]**

**Ex. INTERVAL ‘2 5’ Year To Month = +2-5**

**Ex. INTERVAL ‘50’ Year = +50-0**

**Ex. INTERVAL ‘50’ Month = +4-2**

* Day To Second can provide an interval starting with days and going down to fractional seconds ('1 3:45:10.555555' = 1 day, 3 hours, 45 minutes, 10.555555 seconds). Default precision for days is 2, for fractional seconds is 6. Both can be 0-9. Can simply be day, hour, minute, second or a mix.

**Syntax:**

**Ex. INTERVAL '2 4' Day To Hour = +2 4:0:0**

**Ex. INTERVAL '2 4:1' Day To Minute = +2 4:1:0**

**Ex. INTERVAL '30' Hour = +1 6:0:0**

**CANNOT BE:**

**INTERVAL '2 4' Day To Minute**

**INTERVAL '2 4:1' Minute To Day**

**Privileges/Data Control Language (DCL):**

* Grant statements are only allowed on a **single table at a time**; they can be assigned to **multiple users or roles in a single statement**
* You cannot use **WITH GRANT OPTION** with ROLES!
* System privileges use the **WITH ADMIN OPTION;** you cannot use **WITH GRANT OPTION** with system privileges!
* A privilege which is typically and **OBJECT PRIVILEGE** becomes a **SYSTEM PRIVILEGE** when the word **ANY** is included (e.g. CREATE TABLE vs CREATE ANY TABLE); this distinction grants the correlated privilege for use outside of the user’s schema
* Revocation of object privileges which were granted to other subordinate users has a cascading effect (e.g. Bob grants CREATE TABLE WITH GRANT OPTION to Jimmy, Jimmy grants the same to Samantha; when Bob revokes that privilege from Jimmy, it is also revoked from Samantha)
* If a privilege granter is dropped from the db, any object privileges they granted **DO NOT** cascade
* Revocation of system privilege which was granted WITH ADMIN OPTION **DOES NOT** cascade
* To give a privilege to everyone you would say PUBLIC;

**Transactions and Commits:**

* ROLLBACK with SAVEPOINT doesn't do an implicit commit; ROLLBACK with no modifiers issues implicit commit
* All DDL statements issue implicit commit **before and after** executing;
* When a DBA issues TRANSACTIONAL with the SHUTDOWN command, this allows user to commit their pending transactions on their own prior to shutdown;
* **Typically** an implicit commit happens when a user exits normally, but this configurable on a per app basis, **so it may not necessarily be true**
* Alter Table Is DDL, so implicit commit; triggers, constraints, and indexes; space can be retained, **but it's not ALWAYS retained, it drops by default**
* Savepoint don't issue commits and can only rollback DML; as soon as DDL or COMMIT is issued, savepoint disappears

**Performance Junk:**

* Between Oracle vs SQL:1999 there are no performance differences; Both syntaxes support cartesian products of two tables; SQL:1999 supports NATURAL JOIN syntax
* Non-equijoin has no syntactical performance adjustments to be made
* The BETWEEN and inequality operators (<,>, etc.) have equal performance
* ‘IN’ and ‘=ANY’ equal performance to a WHERE clause using only OR operators
* If the character length of a field is known and **NOT** variable CHAR is more performant than VARCHAR
* Though TIMESTAMP is more precise than DATE type, the DATE type is more **efficient** for calculations
* Query processing may read data from storage if the required data isn't already in memory

**Miscellaneous Crap:**

* When NLS date format is DD-MON-RR, the server can handle a date in the format of DD-MONTH-YYYY for purposes of date arithmetic, etc.
* Automatic Workload Repository (AWR) provides performance statistics
* USER\_CATALOG and CAT are the same
* All usernames including the DBAs are stored in **data dictionary**; the data dictionary is not created or maintained by DBA;
* Enterprise Manager is available for a database after database creation and can perform basic admin tasks for PDBs by using the EM interface
* TRUNCATE and DROP Partitions both support asynchronous Global Index Maintenance
* CURRENT\_DATE & CURRENT\_TIMESTAMP refer to SESSION; SYSDATE refers to SERVER
* DUAL can be used to display a single row, but multiple columns; It is in the SYS schema; It's default config is a single column "DUMMY" and a single row of the type VARCHAR2