Select Distinct cannot sort an expression not in select SELECT [ALL | DISTINCT] [TOP (expression) [PERCENT] [WITH TIES]] < select_list > [FROM { <table_source> } [,...n]] [WHERE <search_condition>] [GROUP BY { column_expression } [,...n]] [HAVING < search_condition >] [ORDER BY { order_by_expression [ASC | DESC] } [,...n] [OFFSET { offset_count_expr } { ROW | ROWS } [FETCH { FIRST | NEXT } { fetch_count_expr } { ROW | ROWS } ONLY]]]
1. () 2. *, /, % 3. +, - 4. =, >, <, >=, <=, <> 5. NOT 6. AND 7. BETWEEN, IN, LIKE, OR 8. = (Assignment)
Armstrong's Rules – 1. **Splitting rule and Combining rule**. A1, A2,.. An -> B1, B2, Bn == A1,A2,..An ->B1,A1,A2,..An ->Bn **Trivial** Rule -> A,B,C -> A..A,B,C -> B etc. **Transitive Closure Rule**: If A1, A2,.. An -> B1, B2, Bn and B1, B2, Bn -< C1, C2, CN Then A1, A2, AN -> C1, C2, CN Given a set of attributes A1, …, An the closure {A1, …, An}+ = the set of attributes B such that A1, …, An → B. A **superkey** is a set of attributes A1, ..., An s.t. for any other attribute B, we have A1, ..., An → B. A **key** is a minimal superkey.. **If X+ =** all attributes then X is a superkey.
A relation R is in BCNF if and only if for every functional dependency X → A: •X → A is a trivial functional dependency Or •X is a superkey for R. A **UPDATE** statement updates all rows as an atomic operation. **WHAT ARE INDEXES?** Structures to allow quick "look-ups" of the requested rows. Physically use B-Trees, Index keys are used for root and intermediate level pages, Other data in the leaf level pages. CLUSTERED INDEXES – Define how the data is physically stored for a table, the leaf-level pages contain all data, CAN ONLY HAVE ONE CLUSTERED INDEX per table, A table without a clustered index is a "heap" **Nonclustered Indexes** – Can have multiple nonclustered indexes, The leaf-level pages contain references to the rows in the clustered index, and may contain copies of other columns, called "include columns". Both Unique and Filtered Indexes are useful for enforcing uniqueness on only non-null values, and only active or non-"removed" rows. **ACID Properties** •Atomicity - All or nothing will succeed •Consistency - Ensures one valid state to another •Isolation - Concurrent transactions are independent •Durability - Committed changes are not lost. **Isolation Levels –** Read Uncommitted – no data consistency, uses no shared locks. Read Committed – Prevents dirty reads. Repeatable Read – prevents dirty and non-repeatable reads. Serializable – prevents all consistency issues, including phantom rows. SNAPSHOT ISOLATION LEVEL – Same level of consistency as serializable, writers don't block readers, readers don't block writers, much improved concurrency. Better consistency and concurrency at a cost. **Transaction Log,** Changes are recorded in the transaction log before written to disk, The log is necessary to provide durable and atomic transactions. A and D of ACID. A subquery can have an ORDER BY but it is only used when a TOP clause is specified. In evaluating the functional dependencies and closures of a relation R, which of the following may be true of R if R is in Boyce-Codd Normal Form? **One** or more of its functional dependencies may be a trivial functional dependency, **For** one or more of its functional dependencies X -> Y, X may be a superkey for R. **For** one or more sets of its attributes X, the closure of X is X itself, i.e. X+ = X. **For** one or more sets of its attributes X, the closure of X contains all attributes, i.e. X+ = [all attributes].**Which of the following are true of a transaction log?** Circle all that apply: **Changes** are recorded in the transaction log before written to disk, **The** transaction log is necessary to provide durable transactions, **the** transaction log is necessary to provide atomic transactions. **According** to the standard, which of the following are possible result types of a subquery? **Scalar value, Table Valued, Multi-valued.**

7. **5 points** – Indicate the logical processing phases of an INNER JOIN.

____ Adds rows from the preserved table that did not match the join predicate
1 Produces a Cartesian Product.
____ Filters out rows that have a NULL mark for its key.
2 Filters rows according to the join predicate.
____ Applies the rows in the right table to each row in the left table.

8. **5 points** – Indicate the logical processing phases of an OUTER JOIN.

3 Adds rows from the preserved table that did not match the join predicate
1 Produces a Cartesian Product.
____ Filters out rows that have a NULL mark for its key.
2 Filters rows according to the join predicate.
____ Applies the rows in the right table to each row in the left table.

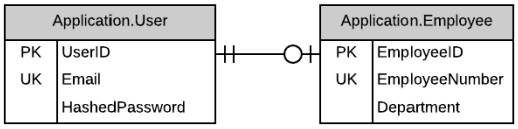
9. **5 points** – Indicate the logical processing phases of a CROSS JOIN.

____ Adds rows from the preserved table that did not match the join predicate
1 Produces a Cartesian Product.
____ Filters out rows that have a NULL mark for its key.
____ Filters rows according to the join predicate.
____ Applies the rows in the right table to each row in the left table.

10. **10 points** – Indicate the logical processing order of a query by writing 1 – N
Write a "1" next to the element that's logically processed first, "2" for the se
may need to repeat a number.

5 SELECT 6 DISTINCT 8 TOP
1 FROM
2 WHERE
3 GROUP BY
4 HAVING
7 ORDER BY
7 OFFSET ... FETCH ...

25. **5 points** – Review below tables, noting the relationship shown between the two. What would need to be done to the tables to implement the relationship shown?



Circle the letter of the **most complete** solution below. Circle only one answer.

a. 1) Add a column named `UserID` to the `Application.Employee` table.
2) Create a foreign key constraint on the new column `UserID` with it referencing `UserID` of `Application.User`.
3) Create a unique constraint on the new column `UserID` of `Application.Employee`.

b. 1) Add a column named `EmployeeID` to the `Application.User` table.
2) Create a foreign key constraint on the new column `EmployeeID` with it referencing `EmployeeID` of `Application.Employee`.
3) Create a unique constraint on the new column `EmployeeID` of `Application.User`.

c. 1) Add a column named `UserID` to the `Application.Employee` table.
2) Create a foreign key constraint on the new column `UserID` with it referencing `UserID` of `Application.User`.

d. 1) Add a column named `EmployeeID` to the `Application.User` table.
2) Create a foreign key constraint on the new column `EmployeeID` with it referencing `EmployeeID` of `Application.Employee`.

e. 1) Add a column named `UserID` to the `Application.Employee` table.
2) Create a unique constraint on the new column `UserID` of `Application.Employee`.

11. Non-clustered indexes provide significant performance gain when querying because they utilize a hash table.

True False

12. You may only have one clustered index on a table.

True False

13. The READ COMMITTED transaction isolation level prevents dirty reads.

True False

14. The SERIALIZABLE transaction isolation level provides the highest degree of consistency.

True False

15. The READ UNCOMMITTED isolation level provides the highest degree of concurrency.

True False

16. The SNAPSHOT isolation level provides the same degree of consistency as SERIALIZABLE.

True False

17. Data modifications are written to the transaction log as part of a CHECKPOINT.

True False

18. A relation is in Boyce-Codd Normal Form if and only if, for every set of attributes X, $X^+ = X$ or $X^+ =$ [all attributes in the relation].

True False

19. A decomposition of a relation into Boyce-Codd normal form will always preserve dependencies.

True False

20. A relation R(A,B,C) with functional dependencies {A,B} → C and C → B is in 3rd Normal Form.

True False

5. **5 points** – Which of the following is the correct column expression to add to the SELECT clause in the above query that will provide the number of credits. A transaction is a credit if its amount is greater than zero (Amount > 0.00). Circle only one answer.

a. `COUNT(*)`
b. `COUNT(IIF(T.Amount > 0.00, T.TransactionId, 0))`
c. `SUM(IIF(T.Amount > 0.00, T.Amount, NULL))`
d. `COUNT(T.TransactionId)`
e. `COUNT(IIF(T.Amount > 0.00, T.TransactionId, NULL))`

1. **5**

14. Indexes in a database provide significant performance gain when querying because they are constructed using a B-Tree.

True False

15. You may only have one non-clustered index on a table.

True False

16. Indexes can be used to enforce uniqueness on the key columns.

True False

17. The READ UNCOMMITTED isolation level allows dirty reads.

True False

18. The READ COMMITTED transaction isolation level provides the highest degree of consistency.

True False

19. The SNAPSHOT isolation level provides the same degree of consistency as SERIALIZABLE.

True False

20. READ COMMITTED, REPEATABLE READ, and SERIALIZABLE isolation levels control varying levels of consistency through shared locks.

2. **5**

True False

21. In a trigger, the old (or prior) values of an UPDATE statement are available in the UPDATED virtual table.

True False

22. A self-contained subquery is logically evaluated once with its result used by the outer query.

True False

23. A correlated subquery is logically evaluated once for every row in the outer query.

True False

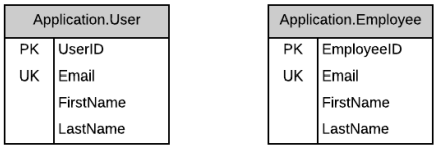
b. It's clear the expected result is one row for each account, however, the result of the joins in the `FROM` clause will give more than one row for each account.

c. The column in the subquery is missing an alias, which is a syntax error.

d. The subquery is performing aggregation, but there is no `GROUP BY` clause.

e. The subquery for `AccountBalance` is incorrect and will provide a value different than the balance for each account.

points – Using the two tables as defined in the diagram below, we want a query that will return sers that are not employees. That is, we want rows that exist in the table `Application.User` **ut not** in the table `Application.Employee`. A given person exists in both tables if and only if ıe value for the column `Email` in `Application.User` also exists as a value in the column `Email` f `Application.Employee`.



ircle the letters of the correct solution(s) below. There may be more than one answer, so circle l that apply.

```
SELECT U.Email
FROM [Application].[User] U
UNION ALL
SELECT E.Email
FROM [Application].[Employee] E;
```

```
SELECT U.Email
FROM [Application].[User] U
EXCEPT
SELECT E.Email
FROM [Application].[Employee] E;
```

```
SELECT E.Email
FROM [Application].[Employee] E
EXCEPT
SELECT U.Email
FROM [Application].[User] U;
```

d.
```
SELECT U.Email
FROM [Application].[User] U
   LEFT JOIN [Application].[Employee] E ON E.LastName = U.LastName
      AND E.FirstName = U.FirstName
WHERE E.EmployeeID IS NULL;
```

e.
```
SELECT U.Email
FROM [Application].[User] U
   LEFT JOIN [Application].[Employee] E ON E.Email = U.Email
WHERE E.EmployeeID IS NULL;
```