CS2102 Finals Cheat Sheet

SQL Queries

CREATE: INSERT INTO <table>(col1, col2 …) VALUES(value1, value2…)
READ: SELECT * FROM <table> WHERE <condition>
UPDATE: UPDATE <table> SET col1 = value1, col2 = value2… WHERE <condition>
DELETE: DELETE FROM <table> WHERE <condition>

- If no condition specified from delete, all table entries deleted

GROUP BY <col1>, <col2>,… - Groups the result by the combination of columns
ORDER BY <col1>, <col2>, … - Orders the result by the first column and resulting groups by the next columns recursively
Both DISTINCT and GROUP BY removes duplicates, but DISTINCT preferred.
However, in some scenarios, no choice but to used GROUP BY

Aggregate Functions

- Uses the GROUP BY operation to perform aggregates on - performs the aggregate function for each group formed using group by.
- If no group by is specified, whole resultant table is taken as single group
- Aggregate functions cannot be used in WHERE clause since it is before GROUP BY, groups have not yet been resolved.

MAX(col): Finds the maximum value of the values in the specified column
AVG(col): Finds the average value of the values in the specified column
SUM(col): Finds the total value of the values in the specified column
COUNT(col): Finds the number of occurrences of values in the specified column. If DISTINCT is used, only unique values are counted.

Set Operations

Set operations automatically removes duplicates
<Q1> UNION <Q2>: Takes all entries from the result of two queries
<Q1> INTERSECT <Q2>: Takes the common entries from the result of two queries
<Q1> EXCEPT <Q2>: Takes the entries that are in Q1 but not in Q2

- Union Compatibility - For set operations to be used, the 2 queries must return results with the same number, order and data types
- Union useful for multiple - condition(<condition1> OR <condition2>), or upper/lower bound(X < 2 AND X > 1)

JOIN: <table1> <TYPE> JOIN <table2> ON <condition1> <TYPE> JOIN <table3> ON <condition2> … - Joining multiple tables together
<TYPE> - *NATURAL JOIN:* Joins tables on values in columns with the *same name*

- For natural join, no condition needs to be specified

INNER JOIN: Joins tables on the condition specified, and returns results where the left query has a corresponding right query
OUTER JOIN: Joins tables on the condition specified and returns rows of the original tables joined on the corresponding values, giving NULL where there are no corresponding entries.

- If LEFT OUTER JOIN specified, all rows from the LEFT query are selected with corresponding joined entries from the RIGHT query matched to each entry, NULL otherwise, while RIGHT OUTER JOIN gives all rows from the RIGHT query with corresponding entries joined from LEFT query, NULL OTHERWISE. By default, FULL OUTER JOIN is specified.
- ON condition specified usually uses <table1>.PK = <table2>.FK condition to ensure data consistency and lossless join.

Nested Queries - Useful Cases

Where X <condition> ALL <case> - …WHERE <table>.<col> >/=/< ALL SUBQUERY
Where X <condition> AT LEAST <case> - …WHERE <table>.<col> >/=/< ANY SUBQUERY
Where X is <condition> - …WHERE EXISTS (SUBQUERY)
Where X is one of the options - …WHERE X = ANY(SUBQUERY)

- Equivalent to WHERE X IN <set>
- Usually subquery is correlated with the attribute X
- IN & = ANY are synonymous in most cases and can be used in place of each other
- NOT IN and NOT EXISTS are also synonymous, but NOT IN may result in NULL entries while NOT EXISTS only returns true or false.
  - e.g SELECT … WHERE <column> NOT IN (SELECT …{may be empty})

Keys

Primary Key - Minimal set of attributes that uniquely identifies each entry in the table chosen from a set of candidate keys. Can only have one per table
SQL: <attribute> <type> PRIMARY KEY(Single Attribute Key),
PRIMARY KEY(<attribute1>, <attribute2>,…) (Composite Key)
Candidate Key - Minimal set of attributes that uniquely identifies each entry in the table. Can have multiple a table
Foreign Key - A column that is a foreign key needs to point to a valid existing value in a column in another table that it references.

- Foreign Key only means if the column takes a value, it must be an existing column in the referenced column. But the Foreign Key can be NULL(Take no value)

SQL: FOREIGN KEY(<attribute1>, <attribute2>,…) REFERENCES <table>(<attribute1>, <attribute2>…)
Super Key - Set of attributes that uniquely identifies each entry in the table.

Entity-Relationship

Weak Entity - Entity where attributes from primary key depends on the values of columns in another table (has foreign key relationship with column in another table)
Participation constraints - *(x, y)* where x is the minimum participation, and y is the maximum participation in a relationship
(1, X) - Compulsory participation (0, X) - Optional participation
(X, 1) for all entities entail a one-to-one participation for entities involved since entities can participate maximum once, and only has one mapping to another entity
(X, N) for all entities entail a many-to-many participation for entities involved
(X, N) for one entity, (X, 1) for another entail a one-to-many relationship involved

Relational Calculus

$A \rightarrow B \equiv \sim A \vee B \equiv \sim (A \wedge \sim B)$ - If A, then B
$\sim \forall X \ f(X) \equiv \exists X \ \sim f(X)$ - There exists X such that $f(X)$ is not fulfilled/ Not all X fulfils $f(X)$
$\sim \exists X \ f(X) \equiv \forall X \ \sim f(X)$ - There does not exist X such that $f(X)$ is fulfilled/ For all X, $f(X)$ is not fulfilled

- If a attribute in the statement needs to take 2 values → ∧ operator will return empty result cause attribute cannot fulfil two different values at once
  - Need to use two different attribute sets → each one fulfilling a different value

Domain Relational Calculus

$\{ < X_1, X_2, .. > \ | \ \exists / \forall \ Y_1, Y_2, \ldots f(X_1, .., Y_1, \ldots)\}$

- The set $< X_1, X_2, \ldots >$ entails a set of attributes that is required, and each variable in the statement is an attribute
- $< X_1, X_2, \ldots > \in E \equiv E(X_1, X_2, \ldots)$ where E is an entity is saying that the combination of attributes $< X_1, X_2, \ldots >$ is present in an entry in the table for E

Tuple Relational Calculus

$\{ \ T \ | \ \exists / \forall \ S'_1 \in S_1, \ S'_2 \in S_2, .. f(T, \ S'_1, \ S_2 \ , \ldots)\}$

- T in the variable list is a tuple of attributes required, and each S' is a row entry in the corresponding table S
- Attributes of tuples can be accessed and compared using $S'.x$ where x is an attribute of the tuple S'

Functional Dependencies

For any attributes X, Y, X →Y is a functional dependency implying that attribute Y depends on attribute X, and {S} → {T} where S and T are attribute sets determine the dependency of set T on set S

- Functional dependency used to identify the relationships between attributes and prevent/remove insertion/deletion/update anomalies
  - i.e NULL values, extra columns, duplicate columns

An FD is known as a *trivial FD* if for X → Y, Y ⊂ X OR Y is an empty set(Y = ∅)

Armstrong's Axioms

1. Reflexivity: If $X \subset Y$, then $X \rightarrow Y$
   1.1. A set with more attribute that contains another set with less attributes is a superset of that set ($XY \subset XYZ$)
   1.2. Can be used in conjunction with transitivity to create relations to subsets ($\underline{X \rightarrow YZ}$, $YZ \rightarrow Y$ since $\underline{Y \subset YZ}$, then $X \rightarrow Y$ by transitivity)
2. Augmentation: If $X \rightarrow Y$, then $XZ \rightarrow YZ$
   2.1. FDs can be augmented using attributes already inside the FD ($X \rightarrow Y$, then $X \rightarrow XY$ by augmentation with X)
3. Transitivity: If $X \rightarrow Y$, and $Y \rightarrow Z$, then $X \rightarrow Z$

A set of FDs is said to be *sound* if each FD in the set can be derived using Armstrong's Axioms
A set of FDs is said to be *complete* if eacf of Armstrong's Axioms can be derived using the set of FDs

Useful Axioms

- *Weak Augmentation* - if $X \rightarrow Y$, then $X \cup Z \rightarrow Y$
- *Union* - if $X \rightarrow Y, Y \rightarrow Z$, then $X \rightarrow YZ$
- *Decomposition* - if $X \rightarrow YZ$, then $X \rightarrow Y$ and $Y \rightarrow Z$
- *Composition* - if $X \rightarrow Y$, $A \rightarrow B$, then $XA \rightarrow YB$
- *Pseudo - Transitivity* - if $X \rightarrow Y$, $YZ \rightarrow W$, then $XZ \rightarrow W$

Closure

Closure of a set of FDs F is the set of all FDs that F entails, derived using Armstrong's Axioms, known as F+

- Two sets of FDs, F and G, are equivalent if they have the same closure($F^+ \equiv G^+$)

Closure of a set S of attributes is the maximum set of attributes S+, that is derived as a consequence of the set of FDs F on S - S+ is the closure of S with respect to F

- Two sets of FDs, F and G, are equivalent if for each $X \rightarrow Y \in F$, then $Y \subset X^+_G$
  Similarly, $X \rightarrow Y \in G$, then $Y \subset X^+_F$
  - i.e for each FD in X, attributes Y entailed by X in F must be in the attribute closure of X in G, meaning each X in G entails all Y entailed by X in F

Minimal Cover

A set of dependencies is minimal if and only if:

1. Every right hand side is a single attribute → Reduce the right hand side
2. For each FD $X \rightarrow Y \in F$, there should not be a FD $X' \rightarrow Y$, where X' is a proper subset of X such that $[F - (X \rightarrow Y)] \cup (X' \rightarrow Y) \equiv F$ → Reduce left hand side by removing redundant attributes in X for each $X \rightarrow Y \in F$
3. For each remaining FD $X \rightarrow Y \in F$, $[F - (X \rightarrow Y)]) \neq F$ → Remove any FD $X \rightarrow Y \in F$ which can be derived from other FDs in F
4. (Extended Minimal Cover) Perform union on each FD $X \rightarrow Y \in G$ where G is minimal, if X if equal.

- For each $X \rightarrow Y \in F$, and each $X \in F$, if X only appears on the left hand side, it must be a prime attribute(part of superkey).

Normal Forms

Prime Attribute - attribute that is part of any candidate key in R

Second Normal Form(2NF)

For each FD $X \rightarrow Y \in F$, X should not be a proper subset of a candidate key - There should not be partial dependency for each FD $X \rightarrow Y \in F$
For each $X, Y : X, Y \subset R, X \rightarrow Y \in F^+$ then

- The FD $X \rightarrow Y$ is trivial - $Y \in X$
- X should not be a proper subset of a candidate key for R, $X \not\subset S$, S is a candidate key for R
- Y should be a prime attribute

Third Normal Form(3NF)

For each FD $X \rightarrow Y \in F$, X should not be a non-prime attribute if Y is a non-prime attribute - No transitive dependency on non-prime attributes

For each FD $X, Y : X, Y \subset R,\ X \rightarrow Y \in F^+$, then
- The FD $X \rightarrow Y$ is trivial - $Y \in X$
- R is in second normal form 2NF
- X should be a super key
- Y should be a prime attribute

Boyce-Codd Normal Form(BNCF)

Stricter form of 3NF, for BCNF for each $X \rightarrow Y \in F$, then X must be a prime attribute - No dependencies on any non-prime attribute.

For each FD $X, Y : X, Y \subset R,\ X \rightarrow Y \in F^+$, then
- The FD $X \rightarrow Y$ is trivial - $Y \in X$
- R is in third normal form 2NF
- X should be a super key

For third normal form, FDs where prime attributes depend on non-prime attributes are allowed to give rise to additional candidate keys, but in BCNF there can not be any dependencies on non-prime attributes

Normalization

Decomposition

A relation schema F can be further decomposed into smaller fragments F_i such that the attributes of F_i is a subset of F, such that $(\cup\ F_i) \equiv F$

A decomposition is lossless if performing a NATURAL JOIN on all of the decomposed fragments F_i returns the original table
- For F_1 and F_2 decomposed from F, the decomposition is lossless if $F_1 \cap F_2 = X,\ X \rightarrow F_2$ meaning primary key of one of the decomposed relations must be a common attribute in the other relation

A decomposition is lossy if:
- NATURAL JOIN removes some of the original entries
- NATURAL JOIN creates new entries using cartesian product, including NULL

Dependency Preserving

Each decomposed relation R_i inherits a set of FDs F_i from the original set of FDs F
Known as projected functional dependencies

A decomposition is dependency preserving if $(\cup\ F_i)^+ \equiv F^+$, if the closure of the combined projected FDs is equivalent to closure of the original set of FDs
- Projected FDs for each decomposed relation R_i may be derived from F^+ instead

BCNF Decomposition

Algorithm: While all relations R_i in R are not in BNCF:

For each $F_i = X \rightarrow Y$ projected by R_i that violates BNCF for $R_i \rightarrow$ Decompose the relation scheme R_i using the FD F_i

Let $R = (R - \{R_i\}) \cup \{X^+, (R_i - X^+) \cup X\} \rightarrow$ Let the new relation scheme decomposed by F_i be X^+, and let the key of X^+, X be in the original relation scheme violated R_i after removing X^+ to ensure lossless decomposition.
- Can always produce lossless, but not always dependency preserving decomposition that is in BCNF form using this algorithm

3NF Synthesis

Starting with a new empty set S, with the original relation scheme R:
Compute the minimal cover of the set of FDs of R being F, let this be F^+
For each $F_i = X \rightarrow Y$ that is in F^+:

If there is no relation scheme(decomposed table) in S that contains the set of attributes $X \cup Y$:

Create a new relation scheme with $X \cup Y$ as it's attributes, inheriting the projected FD $F_i = X \rightarrow Y$

If no relation scheme contains a set of attributes that contains a candidate key for R, create a new relation scheme containing all the attributes in a candidate key for R
- Always able to produce lossless and dependency preserving decomposition in 3NF using this algorithm

Relational Algebra

Projection: Keeps vertical slices of a relation scheme(columns)

$\pi_L(R) = \{t \mid \exists\, t_1\, (t_1 \in R \wedge (t\,.\,A = t_1\,.\,A)_{A \in L})\}$

Selects only the attributes A from a table if A belongs to a list of wanted attributes L
- Projection removes duplicates → emulates group by/distinct clauses

Selection: Selects tuples of a given relation scheme verifying a condition(rows)

$\sigma_c(R) = \{t \mid t \in R \wedge c\}$ where c is a given condition applied on tuples

Set Operations:

Tuples involved in set operations must be union compatible.

Union: $R_1 \cup R_2 = \{t \mid t \in R_1 \vee t \in R_2\}$
- Take the results from both relation schemes

Intersection: $R_1 \cap R_2 = \{t \mid t \in R_1 \wedge t \in R_2\}$
- Take the results common to both relation schemes
- Must be used in cases where one attribute needs to fulfil multiple criteria/values, cannot use $\{t = x \wedge t = y\}$ in condition as t must fulfil 2 conditions at the same time → Impossible, will return empty set

Asymmetric Difference: $R_1 - R_2 = \{t \mid t \in R_1 \wedge \sim (t \in R_2)\}$
- Takes the tuples that is in the first, but not in the second relation scheme

Cartesian Product:

$R_1 \times R_2 = \{t \mid \exists t_1\, \exists\, t_2\, (t_1 \in R_1 \wedge t_2 \in R_2 \wedge t\,.\,A_1 = t_1\,.\,A_1 \wedge t\,.\,A_2 = t_2\,.\,A_2)\}$
- Takes all possible combinations of the tuples from R_1 and R_2
- Can be used to produce a joined table which have a selection condition applied

Joins:

θ - Join: $R_1 \bowtie R_2 =$
$\{t \mid \exists t_1\, \exists\, t_2\, (t_1 \in R_1 \wedge t_2 \in R_2 \wedge t\,.\,A_1 = t_1\,.\,A_1 \wedge t\,.\,A_2 = t_2\,.\,A_2 \wedge c)\}$
$= \sigma_c(R_1 \times R_2)$
- Performs a cartesian product and joins tuples that fulfil a condition c

Equi-Join: $R_1 \bowtie_{E(A_1.a = A_2.a, \ldots)} R_2$
- Joins two tuples based on an equality condition *E (*joins the tuples only if the values in the equality condition match the value in attribute in the other relation)
- Projects only one of the attributes compared in equality condition, since the second one is redundant(due to equality)

Natural Join: $R_1 \bowtie_N R_2$
- Stricter form of the equi-join: joins the tuples based on equality of values of attributes with the same name

Renaming: $\rho\,(R'(A_1 \rightarrow A_1',\ A_2 \rightarrow A_2', \ldots),\ R)$
- Produces a new instance of relation R' that has the same instanced attributes and values as R with each attribute $A_i' \in R'$ having the same value as $A_i \in R$
- Can be used to rename multiple instances of relation scheme A to A_i for use in relational algebra statements
- $\rho\,(R',\ R)$ directly renames R to R' without changing any of the names of attributes - useful for using multiple of the same table in a statement for differentiation.

Division:
- $\dfrac{R_1}{R_2}$ produces the set of attributes x such that for each set of attributes $y \in R_2$, there exists a combination$\{x, y\} \in R_1$
  - Gives all x that has a combination with every $y \in R_2$ in R_1
- Useful in scenarios such as find all A that has B where A , B are attribute sets of $R_1,\ R_2$ respectively

Unsafe Query → Query that returns an infinite number of results
- e.g $\{E \mid \sim (E \in X)\}$

- MAX aggregate using relational algebra: SELECT whole set of attributes wanted from R MINUS cartesian product of R with itself, for values of the second R bigger than the first → $\pi_A(R) - \pi_A(\sigma_{R2.v > R1.v}(\rho(R1,\ R) \times \rho(R2,\ R)))$