Cheatsheet:

- “Tuples” in terms of tables, usually mean rows.
- When two relations/tables have the same “arity,” it means they have the same number of attributes/columns.
- In order for attributes to be compatible, they must have the same Domain Type.

❖ **Select**: $\sigma_{predicate}(r)$ - Selects the tuple that meets the predicate condition from table *r*. For example, $\sigma_{deptName = "Psysics"}(Instructor)$would choose every tuple in the *Instructor* table that has the value of deptName be Physics. A.k.a., choose every Physics instructor.

❖ **Project**: $\Pi_{Attribute\ 1,\ Attribute\ 2,...,\ Attribute\ (n)}(r)$ - Selects the attribute columns and gets rid of the unlisted ones. For example, if you had a table, let’s say *Instructor* again, that has the columns labeled ID, name, deptName, and salary, then $\Pi_{ID,\ name,\ salary}(Instructor)$ would create a subset (subtable) with all the information of the parent table, but without deptName because it was not specified.

❖ **Union**: $r \cup s$- Takes the union of tuples of two relations that have compatible attributes and arity. For example, if there is a table of all classes offered at IIT called *Classes*, and you wanted to find all the classes during the fall and spring semesters. You could select all classes with the semester of fall, then select all classes with the semester of spring, then use the union operator to join them. It would look like this:

$\sigma_{Semester = "Fall"}(Classes) \cup \sigma_{Semester = "Spring"}(Classes)$. Plus, if you just wanted one specific attribute of this paring, say the course_ID, you just need to add a projection of course_ID to the front of the relation:

$\Pi_{courseID}(\sigma_{Semester = "Fall"}(Classes) \cup \sigma_{Semester = "Spring"}(Classes))$

- **Set Difference**: $r - s$ - <u>Removes the available tuple in *s* from *r*</u>. *r* and *s* must have the same arity and their attributes must be compatible. For example, going back to the *Classes* table, if you want all the classes taught in the fall semester, but not the ones that are taught in the spring semester, or in both, the following is applicable:

  $\sigma_{Semester = "Fall"}(Classes) - \sigma_{Semester = "Spring"}(Classes)$. Unlike just

  $\sigma_{Semester = "Fall"}(Classes)$, this would get rid of the tuples of classes that are both in the fall and spring semester, leaving fall exclusive classes. A.k.a. No classes that are offered in both Fall and Spring would be in the created relation.
- **Cartesian Product:**$r\ x\ s$- <u>Creates a relation with every possible tuple combination of *r* and *s*.</u> Must rename some attributes if the attribute in one column of one relation matches one of the other relation.
- **Rename**: $\rho_x(E_1)$ - Renames to x, whatever the relation E_1 evaluates to.
- **Domain type**: varchar (maxSize), char(fixedN), int, smallint, numeric(p,d), float(n) precision at least n digits.

Create table r (A1 D1, A2 D2, ..., An Dn,) (integrity‑constraint: not null, primary key, foreign key references r1),

Add Tuple to Table: **insert into** *table* **values** (a1,a2,…an)

UPDATE *table* SET *attribute = attribute - numericalValue* WHERE *tuple meets predicate*

UPDATE account SET balance = balance -100 WHERE accID = 100;

Drop table: Table allows remove a relation;

DELETES FROM *table* WHERE *tuple meets predicate*

DELETES FROM instructor WHERE dept_name = “Finance”;

LIKE '% substring %' - % character matches any substring within the character.

Set operator union, intersect, except (equivalent to set difference)   f

Alter table add AD allow adding more attribute, alter table drop D removes attribute

select A1, A2, ..., An , *allows selection of tuple of attributes A1, A2, ..., An*

from r1, r2, ..., rm*, the cartisean relation r1, r2, ..., rm*

where P *: where predicate (condition) P is true*

With name as EXPR

group by allow aggregate function (avg, max, min, sum, count)

not in R return / update tuple not in R with clause provi where predicate (condition) P is true

des a way of defining a temporary view whose definition is available only to the query in which the w

ith clause occurs

Join operations take two relations and return as a result another relation as a join on (using) a commo

n attribute(s).

Rule 1) Strong entity E: Create relation with attributes of E , Primary key is equal to the PK of E

Rule 2) Weak entity W identified by E through relationship R: Create relation with attributes of W an

d R and PK(E). Set PK to discriminator attributes combined with PK(E). PK(E) is a foreign key to E.

Rule 3) Binary relationship R between A and B: one‑to‑one: If no side is total add PK of A to as foreign key in B or the other way around. Add any attributes of the relationship R to A respective B. If one side is total add PK of the other‑side as foreign key. Add any attributes of the relationship R to the total side. If both sides are total merge the two relation into a new relation E and choose either PK(A) as PK(B) as the new PK. Add any attributes of the relationship R to the new relation E.

Rule 4) Binary relationship R between A and B: one‑to‑many/many‑to‑one. Add PK of the “one” side as foreign key to the

“many” side. Add any attributes of the relationship R to the “many” side.

Rule 5) Binary relationship R between A and B: many‑to‑many: Create a new relation R. Add PK’s of A and B as attributes

+ plus all attributes of R. The primary key of the relationship is PK(A) + PK(B). The PK attributes of A/B form a foreign key *to A/B*

Rule 6) N‑ary relationship R between E1 … En. Create a new relation. Add all the PK’s of E1 … En. Add all attributes of R to

the new relation. The primary key or R is PK(E1) … PK(En). Each PK(Ei) is a foreign key to the corresponding relation.

Rule 7) Entity E with multi‑valued attribute A. Create new relation. Add A and PK(E) as attributes. PK is all attributes.

PK(E) is a foreign key.