**Introduction**

* **Superkey**: subset of attributes that uniquely identifies a tuple in a relation
* **Candidate key**: minimal superkey
* **Foreign key**: refers to the primary key in another relation

**Relational Algebra**

* Selection σ, projection π, renaming ρ, union ∪, cross product X, set difference –, inner join ⨝, outer join ⟗

**ER Model**

|  |  |
| --- | --- |
|  |  |
|  | **At most one** |
| **At least one** | **Exactly one** |

**SQL Table**

* **Conditions:** <, <=, =, >, >=, <>, IS (NOT) NULL, IS (NOT) DISTINCT FROM, OR, AND, BETWEEN…AND, IN
* **CREATE TABLE** Table (attribute TYPE, )
* **INSERT INTO** Table [(attributes, )] VALUES (values, )
* **DELETE FROM** Table [WHERE condition]
* **UPDATE** Table SET (new value) [WHERE condition]
* **Constraints:** NOT NULL, UNIQUE
* **PRIMARY KEY** (attribute)
* **FOREIGN KEY** (attribute) **REFERENCES** Table (attribute) [ON DELETE <action>] [ON UPDATE <action>]
* NO ACTION/ RESTRICT/ CASCADE/ SET DEFAULT/ SET NULL
* **CHECK** condition
* **NOT DEFERRABLE | DEFERRABLE INITIALLY DEFERRED | DEFERRABLE INITIALLY IMMEDIATE**
* **ALTER TABLE** Table **ADD|DROP COLUMN** attribute
* **ALTER TABLE** Table **ADD|DROP CONSTRAINT** attribute (constraint)
* **DROP TABLE** [IF EXISTS] Table [CASCADE]
* **CAST** (attribute AS TYPE)

**SQL Query**

* **SELECT** [DISTINCT] attributes FROM Table [WHERE condition] [ORDER BY attribute ASC|DESC] [LIMIT k [OFFSET o]]
* **UNION | INTERSECT | EXCEPT** [ALL -> does not remove dups]
* Table1 m **JOIN** Table2 n ON m.attribute = n.attribute
* **ANY | ALL | EXISTS** (subquery)

**SQL Aggregation**

* **MIN | MAX | AVG | COUNT | SUM**
* Attribute -> non-null, \* -> all including null, DISTINCT attribute -> distinct non-null
* Returns null for min/max/avg/sum if does not exist
* [HAVING] **GROUP BY**
* Selected attribute must be either in the group by clause
* Be an input in the aggregate function
* Be a primary key
* **CASE** WHEN condition THEN result1 (…) ELSE result2
* **COALESCE** -> returns first non-null
* **NULLIF** (v1, v2) -> returns null if v1 = v2
* **WITH** **[RECURSIVE]** CommonTableExpression **AS** (query) SELECT …

**Functions**

* Returns values
* **CREATE OR REPLACE FUNCTION** f ([IN | OUT] attributes) RETURNS output AS $$ (query) $$ LANGUAGE sql
* can return value type (e.g. INT), tuple with the same types as a table (e.g. Table), multiple tuples (e.g. SETOF Table), new tuple (using OUT e.g. returns RECORD), new tuples (e.g. SETOF RECORD), new tuples (e.g. TABLE (attributes TYPE))

**Procedures**

* No return value
* **CREATE OR REPLACE PROCEDURE** p (attribute TYPE) AS $$ (query) $$ LANGUAGE sql
* **CALL** p (…)

**SQL Control Structures**

* **DECLARE** var TYPE
* **IF** … THEN … [ELSIF … THEN …] [ELSE …] END IF
* **[WHILE … | FOR … IN …]** **LOOP** … END LOOP
* **[REVERSE] 1..10** -> range from 1 to 10
* DECLARE curs **CURSOR** FOR (query); r RECORD;
* **FETCH** [PRIOR | FIRST | LAST | ABSOLUTE n FROM] curs INTO r
* **EXIT WHEN NOT FOUND**

**SQL Triggers**

* CREATE OR REPLACE FUNCTION f (…) RETURNS **TRIGGER** AS $$ <trigger\_code> RETURN <return value> $$ LANGUAGE plpgsql
* Return type:

|  |  |  |
| --- | --- | --- |
| Event | New | Old |
| INSERT | ✓ | X |
| UPDATE | ✓ | ✓ |
| DELETE | X | ✓ |

* **CREATE TRIGGER** name [BEFORE|AFTER] event ON Table FOR EACH [ROW|STATEMENT] [WHEN condition] EXECUTE FUNCTION f();
* Triggers are activated in alphabetical order
* **Null return**

|  |  |  |
| --- | --- | --- |
| Event | NULL | Non-NULL |
| BEFORE | Does not proceed with trigger | Trigger proceeds normally |
| AFTER | **NO EFFECT** | **NO EFFECT** |

* **TG\_OP** can be used to find what event was triggered
* FOR EACH STATEMENT ignores NULL returns
* Can fail by **RAISE EXCEPTION** instead

**Functional Dependency**

* X -> Y
* If X1 = X2, then Y1 = Y2
* A functional dependency is **trivial** if Y ⊆ X
* A functional dependency is **completely non-trivial** if Y ≠ Ø and Y ∩ X = Ø
* S is a **superkey** of relation R iff S -> R
* S is a **candidate key** if S -> R and for all T ⊂ S, T is not a superkey of R (aka no smaller superkey)
* **Prime attribute** is an attribute in the candidate key
* **Closure,** , is the set of al FDs logically derived from
* Two sets of FDs are **equivalent** if their closures are the same
* These two sets can be considered **covers** of each other
* The **closure** of a set, S­­+, is the set of all attributes FD on S
* Input: S, , Output: S+

Ω :=  // unused, Γ := S // closure

While X -> Y ∈ Ω and X ⊆ Γ do:

Ω := Ω – {X -> Y}, Γ := Γ ∪ Y

Return Γ

* **Armstrong Axioms**
* **Reflexivity**: (Y ⊆ X) => (X -> Y)
* **Augmentation**:(X -> Y) => (XZ -> YZ)
* **Transitivity**: (X -> Y ^ Y -> X) => (X -> Z)
* (e.g.) Weak augmentation: (X -> Y) => (XZ -> Y)

X -> Y (given)

XZ -> X (by reflexivity of X ⊆ XZ)

XZ -> Y (transitivity), Q.E.D

* A set of FDs is **minimal** iff
* The right-hand side is minimal, X -> {A}
* For X -> {A} there is no Y -> {A} in such that Y ⊂ X
* None of the FDs can be derived from other FDs
* A set of FDs is **compact** iff
* No two FDs with the same left hand side (e.g. ({A} -> {B}, {A}->{C} convert to {A} -> {BC}
* A **compact minimal cover (aka canonical cover)** is compact, minimal, and equivalent to

1. Simplify the RHS
2. Simplify the LHS
3. Remove FDs that can be derived
4. Regroup all FDs with the same LHS (undo step 1) -> skip this step for just minimal cover

**Anomalies**

* Redundant storage, update anomalies, deletion anomalies, insertion anomalies

**BCNF**

* A set of FDs is in **BCNF** iff
* X -> {A} is trivial or
* X is a superkey
* If the table is not in BCNF, **decompose** the table along that FD (e.g. X -> Y)
* R1 = X+
* R2 = (R – X+) ∪ X
* A binary decomposition is a **lossless-join** only if the full outer natural join equals the initial table
* A decomposition is **dependency preserving** if the union of the covers of the decomposed sets are equivalent tot eh cover of the original set of FDs
* BCNF is **lossless but not dependency preserving** (e.g. {A, B}-> {C}, {C} -> {B})

**3NF**

* A set of FDs is in **3NF** iff
* X -> {A} is trivial or
* X is a superkey or
* A is a prime attribute
* **Algorithm**

1. For each X -> Y in the **minimal compact cover**, create a relation X ∪ Y unless it already exists or is subsumed by another relation
2. If none of the relations contains the primary key, create a relation with the primary key

* Note that the final tables may not be in BCNF (the example in the lecture is coincidence)