Definition of Terms

Relations

- Schema: Collection of attributes (table header)
- Degree/arity: number of attributes in the schema/number of columns
- Cardinality: number of rows
- Constraints: Ensuring the validity of the instance
- Relational database schema: A collection of all schemas

Keys

- Superkey: A set of attributes that can uniquely identify rows in a relation
- Key: Minimal superkey; cannot remove extra attributes from the superkey
- Candidate keys: all possible keys
- Prime attribute: an attribute within any of the candidate keys
- Primary key: the chosen candidate key
- Foreign key: If (A, B) refers to (C, D), then (C, D)
    - Must be a candidate key and
    - (a, b or both are NULL) OR ((a, b) exists in (C, D))

NULL

- Unknown value: neither true nor false
- NULL and NULL = NULL; NULL or NULL = NULL
- NULL or FALSE = NULL; NULL and TRUE = NULL
- NULL + x = NULL; NULL > x = NULL
- (NULL == NULL) = NULL; (NULL != NULL) = NULL
- (NULL === NULL) = TRUE (not distinct from); (NULL !== NULL) = FALSE (distinct from)

Relational Algebra

- Selection, sigma ($\sigma_{condition}$): filters rows
- Projection, pi ($\pi_{col1,col2}(relation)$): chooses columns
- Renaming, rho ($\rho_{newName}(relation)$): renames tables/columns

Table Creation

Create table

```
CREATE TABLE table1 (
  col1 INTEGER PRIMARY KEY,
  col2 VARCHAR NOT NULL DEFAULT 'some default',
  col3 INTEGER REFERENCES table2(foreign_col),
  FOREIGN KEY (col1, col2) REFERENCES table3(col1, col2) ON
DELETE CASCADE,
  UNIQUE (col1, col3),
  CHECK (col1 > 0)
);
```

Foreign key actions

- ON DELETE/UPDATE
    - CASCADE : propagates action to all referencing tuples
    - SET DEFAULT : set to default value if possible
    - SET NULL : set to NULL value if possible

Constraints reject on FALSE (accepts NULLs)

Conditions accept on TRUE (rejects NULLs)

Table Modification

Alter

```
ALTER TABLE table1
ADD COLUMN col4 INTEGER,
DROP COLUMN col3,
ADD UNIQUE (col4),    -- table constraint
DROP CONSTRAINT constraint_name;
```

Insert

```
INSERT INTO table1 (col1, col2, col3) VALUES (1, 'New value', 3),
(2, 'Second' , 4);
```

Delete

```
DELETE FROM table1 WHERE col1=1 AND col2='New value';
DELETE FROM table1 WHERE col1 < 0;
```

Update

```
UPDATE table1 SET col2='Alice' WHERE col1=2;
```

Join Operations

Inner join ⋈

```
SELECT * FROM t1 INNER JOIN t2 ON t1.col1 = t2.col1;
```

- Selects records that have matching values

Natural join

```
SELECT * FROM t1 NATURAL JOIN t2;
```

- Selects records that have matching values, no need to specify which column and does not repeat joined column

Left outer join ⟕

```
SELECT * FROM t1 LEFT JOIN t2 ON col1=col2;
```

- Selects all records from left table, and matched records from right table, appends NULL values if no match in right table

Right outer join ⟖

- Opposite of left outer join

Full outer join ⟗

- Similar to other outer joins, puts NULL values on both left and right tables

Set Operations

Union/union all

- Combines the 2 result sets
  - Same number of columns, same or more rows
- UNION ALL: keeps duplicates

Intersect/intersect all

- Selects rows that are the same between 2 tables
  - Same number of columns, same or less rows
- INTERSECT ALL: keeps duplicates

Except/except all

- Selects rows that are in t1 but not t2
  - Same number of columns, same or less rows
- EXCEPT ALL: weird, don't really use it

Querying

Operators

`+ - * / % ^ !`	
`& | << >>`	Bitwise expressions
`|/ ||/ @`	Square root, cube root, abs
`> >= < <= == != or <>`	Comparators
`||`	String concatenation
`char_length`	Number of characters
`substring(col1, start, end)`	Extract a substring
`AND OR NOT IS NULL IS NOT NULL`	

Basic query

```
SELECT DISTINCT t1.col1 AS first, t2.col1 AS second
FROM table1 t1, table2 t2
WHERE t1.col1 = t2.col2
AND t1.col1 > 3;
```

Pattern matching

```
SELECT * WHERE col1 LIKE '_ab%'
```

- _ matches 1 char length
- % matches 0 or more characters
- 'ab' matches exact characters

Null if

```
SELECT NULLIF(col1, 3), col2 FROM table1;
```

- Returns NULL if col1=3, else it returns the value of col1

Coalesce

```
SELECT COALESCE(col1, col2, col3) FROM table1;
```

- Returns the first non-null attribute

Case analysis

- Syntax

```
CASE
    WHEN col1 < 0 THEN 'sm'
    WHEN col1 < 3 THEN 'md'
    ELSE 'lg'
END;
```

- Example in Select

```
SELECT OrderID,
CASE
    WHEN Quantity > 30 THEN 'Quantity is greater than 30'
    ELSE 'Quantity is less than or equal to 30'
END AS Description
FROM Orders;
```

- Example in Order By

```
SELECT Name, Gender
FROM Customers
ORDER BY
    CASE WHEN Gender = 'F' THEN Name END DESC,
    CASE WHEN Gender = 'M' THEN Name END ASC;
```

Aggregate functions

```
SELECT MIN(col1) AS Minimum, MAX(col1) AS Maximum FROM table1;
```

function	If empty	If all NULL	meaning
`MIN(A)`	NULL	NULL	Min value of A
`MAX(A)`	NULL	NULL	Max value of A
`SUM(A)`	NULL	NULL	Sum of all values in A
`COUNT(A)`	NULL	NULL	Count of non-NULL values
`AVG(A)`	0	0	Average of values (sum/count)
`COUNT(*)`	0	N	Number of rows in the table
`AVG(DISTINCT A)`			Average of distinct values in A
`SUM(DISTINCT A)`			Sum of distinct values in A
`COUNT (DISTINCT A)`			Count of distinct, non-NULL values

Grouping (with aggregate function)

```
SELECT * FROM table1 GROUP BY col1, col2;
```

- Divides rows into groups where all values on (col1, col2) are the same; on each group
- Use `HAVING` instead of `WHERE` (if using aggregate functions in conditions)
  - `HAVING` is computed after the `GROUP BY` function; `WHERE` is computed before
- Restrictions
  - Either a candidate key appears in GROUP BY, or
  - For each attribute A in the SELECT statement, either
    - A appears in GROUP BY, or
    - A appears in an aggregate function in SELECT

Exists

```
SELECT * FROM table1
WHERE EXISTS (SELECT 1 FROM table2 WHERE col1=table1.col1);
```

- Exists returns TRUE as long as the result of the query is non-empty (at least 1 row)
- Opposite: NOT EXISTS

In

```
SELECT * FROM table1
WHERE col1 IN (SELECT col1 FROM table2);
```

- Returns TRUE if the value of col1 is <u>in</u> the result of the query

Any

```
SELECT * FROM table1
WHERE col1 < ANY (SELECT col1 FROM table2);
```

- Returns TRUE if the value satisfies the relational operation with <u>any</u> of the results

All

```
SELECT * FROM table1
WHERE col1 < ALL (SELECT col1 FROM table2);
```

- Returns TRUE if the value satisfies the relational operation with <u>all</u> of the results

Sorting

```
SELECT * FROM *
ORDER BY col1 ASC, col2 DESC
OFFSET m
LIMIT n;
```

- Orders by col1 ASC overall, col2 DESC within each equal col1 block
- Shows from the mth result, limits to n results ([m, …, m+n-1] results)

Order of operations

```
FROM, WHERE, GROUP BY, HAVING, SELECT, ORDER BY, LIMIT/OFFSET
```

- Subqueries can contain a reference to a result/alias in a parent statement

Common Table Expressions (CTEs)

```
WITH
    table1 AS (SELECT * FROM …),
    table2 AS (SELECT * FROM …)
SELECT * FROM table1, table2;
```

- Declarative way of computing and storing queries sequentially

Views

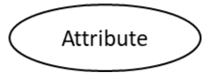
- Used only for querying
- Can be for privacy reasons, can be to compute attributes dynamically, can be for computing ISA (students view where students = undergrads + grads)
- A virtual table that is dynamically computed each time

```
CREATE VIEW view1(first, second) AS
    SELECT col1, col2 FROM table1;
```

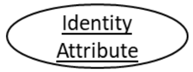
ER Diagrams

Noun

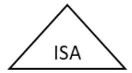
Verb



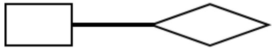
Adjective
(Cannot have its own key)



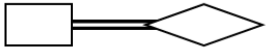
Primary key
Candidate keys not shown



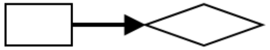
Covering constraint satisfied: in at least 1
Overlap constraint satisfied: can be both



0 or more (unconstrained)

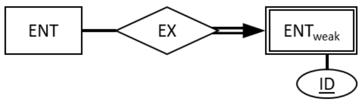


1 or more (total participation)



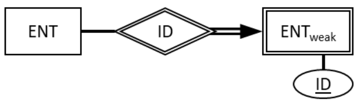
0 or 1 (key)

Exactly 1 (key + total participation)



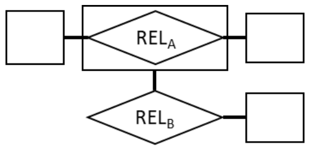
Existential dependency

- Existence of weak entity depends on ENT, but it can be uniquely identified on its own
- Example: child's bank account (unique acc no.)



Identity dependency

- Existence of weak entity depends on ENT, and it cannot be uniquely identified on its own
- ID is a partial key
- Example: a chapter of a book



Aggregation

- Treating a relationship set as an entity set

Functional Dependencies (notation: A - single attribute, a - set)

Definitions

- a → b
  - a uniquely identifies all attributes in b for all valid relation instances
- Triviality of a → b
  - Trivial iff $b \subseteq a$
  - Completely non-trivial if $b \nsubseteq a$ AND $a \cap b = \emptyset$
- Minimal cover
  - Minimal set of functional dependencies, with each in the form a → A
  - At least one FD always exists
  - Method of finding minimal cover
    - Remove attribute redundancy (removing A, (a -A) → b, does not change F)
    - Remove FD redundancy (removing a→b does not change F)

Attribute closure

- a+ = attribute closure of a
  - All the attributes that can be uniquely identified by a
  - a+ = { A in relation R | F |= a → A }

Armstrong's axioms

- Reflexivity
  - a → b for any $b \subseteq a$
  - e.g. ABC → AB; A → A
- Augmentation
  - If a → b, then ac → bc
- Transitivity
  - If a → b AND b → c, then a → c

Extended Armstrong's axioms

- Union
  - If a → b AND a → c, then a → bc
- Decomposition
  - If a → b AND c ⊆ b, then a → c
  - e.g. AB → CDE, so AB → C

Obtaining 3NF DP, LJ preserving decomposition from minimal cover

1. Create schema for each FD in minimal cover (eg A→BC: R(ABC))
2. Create i+1th schema for a key (eg if key is BD: R(BD))
3. Remove schema subsets

Schema Decomposition

Definitions

- Decomposing relation R into fragments such that
  - R = R1 U R2 U … U Rn
- FD projection
  - F[R1] = { all a → b in F+ of R | ab ⊆ R1 }

Properties

- Lossless-join (LJ)
  - Information preserving join; after joining, will obtain original universal relation
  - Criteria for a lossless join decomp (R → R1, R2)
    - If attr(R1) U attr(R2) = attr(R)
    - If attr(R1) ∩ attr(R2) ≠ ∅
    - If common attribute(s) are a superkey/key of either relation
  - For a completely non-trivial FD a → b, { R-b, ab } is a LJ decomp (Corollary 1)
- Dependency preserving
  - After joining, all original FDs are preserved (F[R1] U … U F[Rn] == F)

Normal Forms

Boyce-Codd Normal Form (BCNF)

- No attribute is transitively dependent on any key
  - Every attribute is directly dependent on all keys
- R (or Ri) is in BCNF if, for every a → A in F+ (or F[Ri])
  - a → A is trivial, or
  - a is a superkey
- Theorems
  - Any schema with exactly 2 attributes is in BCNF (Lemma 2)
  - There is always a valid LJ decomp where each fragment is in BCNF (Theorem 3)
  - There exists a schema that is not in BCNF which has no valid LJ + DP decomp such that each fragment is in BCNF (Theorem 4)
- To decompose to BCNF (only guaranteeing lossless)
  - Let a → b be an FD that violates BCNF; decompose R to {R1(ab), R2(R-b)}

Third Normal Form (3NF)

- No non-prime attribute is transitively dependent on any key
  - Every non-prime attribute is directly dependent on all keys
- R (or Ri) is in 3NF if, for every a → A in F+ (or F[Ri])
  - It satisfies BCNF, or
  - A is a prime attribute
- Up to 3NF, it is always possible to get a valid LJ + dep preserving decomp (Theorem 5)