Models and Systems for Big Data Management

Relational Databases

Nacéra Seghouani

Computer Science Department, CentraleSupélec
Laboratoire de Recherche en Informatique, LRI LaHDAK group
nacera.bennacer@centralesupelec.fr

2019-2020

Relational Theory

- ✔ Introduced by E. F. Codd
- ✔ Let $R(\mathcal{A})$ a relation, A_i an attribute $\in \mathcal{A}$ ($i \in [1..n]$), a set of ordered attributes
- ✔ A_i values $\in D_i$ are atomic
- ✔ Let $\mathcal{F}$ a set of functional dependencies $X \rightarrow Y$ defined on $\mathcal{A}$, $X, Y \subseteq \mathcal{A}$
- ✔ Let $\mathcal{T}$ a set of tuples such as $\{(a_1, a_2, \ldots a_n) \in (D_1 x D_2 x \ldots D_n)\}$
  - ✎ Examples:

    ex. $schedule(slot, room, professor, module, group)$;
    $room, slot \rightarrow module, group, professor$
    $professor, slot \rightarrow module, group, room$

    ex. $Beer(name, manufacturer)$;
    $name \rightarrow manufacturer$

Relational Algebra

- ✔ Relational algebra is an algebra where operands are relations.
- ✔ Relational algebra operators are used as a query language for relations in a database.
  - ➤ Union, intersection, and difference: $\cup, \cap, \setminus$.
  - ➤ Selection used for picking certain tuples: σ.
  - ➤ Projection used for picking certain attributes: π
  - ➤ Product (Cartesian) and Join used for composition of relations: $\times, \bowtie$.

Relational Algebra : Selection

- ✔ $R_2 = \sigma_C(R_1)$
  - ➤ C is a condition that refers to attributes of R_1.
  - ➤ R_2 is all tuples of R_1 that satisfy C.

Relation Sells

bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Miller	3.00

Relation $\sigma_{bar=\text{"}Joe's\text{"}}(Sells)$

bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75

Relational Algebra: Projection

- ✔ $R_2 = \pi_X(R_1)$, X is a subset of R_1 attribute's set.
  - ➤ R_2 is constructed by looking at each tuple of R_1, extracting the attributes in X, in the specified order, and creating from them a tuple for R_2.
  - ➤ Eliminate duplicate tuples, if any.

Relation Sells

bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Miller	3.00

Relation $\pi_{beer,price}(Sells)$

beer	price
Bud	2.50
Miller	2.75
Miller	3.00

Relational Algebra: Extended Projection

- ✔ Using the same π_X operator, we allow X to contain arbitrary expressions involving attributes.
  - ➤ For instance arithmetic on number attributes

Relation Prices

beer	price min	price min
Bud	2.50	2.75
Miller	2.75	3.75
Heineken	3.00	4.00

Relation $\pi_{beer,\text{price max}-\text{price min}}(Sells)$

beer	price max − price min
Bud	0.25
Miller	1.00
Heineken	1.00

Relational Algebra: Cartesian Product

- ✔ $R_3 = R_1 \times R_2$
  - Pair each tuple t_1 of R_1 with each tuple t_2 of R_2
  - Concatenation $t_1 t_2$ is a tuple of R_3.
  - If R1 and R2 have attribute A of the same name, use R1.A and R2.A.

Sells			*Bars*	
bar	beer	price	bar	addr
Joe's	Bud	2.50	Joe's	Maple St.
Joe's	Miller	2.75	Sue's	River Rd.
Sue's	Bud	2.50		
Sue's	Miller	3.00		

Sells $\times$ *Bars*				
Sells.bar	Sells.beer	Sells.price	Bars.bar	Bars.addr
Joe's	Bud	2.50	Joe's	Maple St.
Joe's	Bud	2.50	Sue's	River Rd.
Joe's	Miller	2.75	Joe's	Maple St.
Joe's	Miller	2.75	Sue's	River Rd.
Sue's	Bud	2.50	Joe's	Maple St.
Sue's	Bud	2.50	Sue's	River Rd.
Sue's	Miller	3.00	Joe's	Maple St.
Sue's	Miller	3.00	Sue's	River Rd.

Relational Algebra: Theta-Join

✔ $R_3 = (R_1 \bowtie_C R_2) = \sigma_C(R_1 \times R_2)$.

Sells			*Bars*	
bar	beer	price	name	addr
Joe's	Bud	2.50	Joe's	Maple St.
Joe's	Miller	2.75	Sue's	River Rd.
Sue's	Bud	2.50		
Sue's	Miller	3.00		

Sells $\bowtie_{bar=name}$ *Bars*				
Sells.bar	Sells.beer	Sells.price	Bars.name	Bars.addr
Joe's	Bud	2.50	Joe's	Maple St.
Joe's	Miller	2.75	Joe's	Maple St.
Sue's	Bud	2.50	Sue's	River Rd.
Sue's	Miller	3.00	Sue's	River Rd.

Relational Algebra: Natural Join

- ✔ $R_3 = (R_1 \bowtie R_2)$
  - ➤ Natural join connects two relations by equating attributes of the same name, and projecting out one copy of each pair of equated attributes.

Sells			*Bars*	
bar	beer	price	bar	addr
Joe's	Bud	2.50	Joe's	Maple St.
Joe's	Miller	2.75	Sue's	River Rd.
Sue's	Bud	2.50		
Sue's	Miller	3.00		

Sells $\bowtie$ *Bars*			
bar	beer	price	addr
Joe's	Bud	2.50	Maple St.
Joe's	Miller	2.75	Maple St.
Sue's	Bud	2.50	River Rd.
Sue's	Miller	3.00	River Rd.

➤ A join is also an inner join as opposed to an outer join

Relational Algebra: Outer-Joins

✔ Outer join: avoids dangling tuples, tuples that do not join with anything.

Left (outer) join ⟕

Right (outer) join ⟖

Full (outer) join $R_1 ⟗ R_2 = (R_1 ⟕ R_2) \cup (R_1 ⟖ R_2)$

Sells			*Bars*	
bar	beer	price	bar	addr
Joe's	Bud	2.50	Joe's	Maple St.
Joe's	Miller	2.75	Sue's	River Rd.
Sue's	Bud	2.50		
Sue's	Miller	3.00		
Max's	Miller	3.00		

Sells ⟕ *Bars*			
bar	beer	price	addr
Joe's	Bud	2.50	Maple St.
Joe's	Miller	2.75	Maple St.
Sue's	Bud	2.50	River Rd.
Sue's	Miller	3.00	River Rd.
Max's	Miller	3.00	null

Relational Algebra: Set Operators

✔ Let $R_1(\mathcal{A}_1)$ and $R_2(\mathcal{A}_2)$ be two relations such that $\mathcal{A}_1 = \mathcal{A}_2$
Union $\cup$, Intersection $\cap$, Difference $\setminus$ between two relations R_1 and R_2 are similar to set operators.

$S1$			$S2$		
bar	beer	price	bar	beer	price
Joe's	Bud	2.50	Max's	Heinken	4
Joe's	Miller	2.75	Sue's	Bud	2.50
Sue's	Bud	2.50			
Sue's	Miller	3.00			

$S1 \setminus S2$		
bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Miller	3.00

Relational Algebra: Complex Expressions

- ✔ Combine operators with parentheses and precedence rules: Expression trees.
- ✔ Precedence of relational operators: (i) $[\sigma, \pi]$; (ii) $[\times, \bowtie]$; (iii) $\cap$; (iv)$\cup, \setminus$.
- ✔ Give the algebra tree (or expression) for each query. Unary *rename* operator is used to rename a relation (useful to remove ambiguity).

✎ *Find the names of beers sold by "Sue's" bar*

✎ *Find the names of all the bars that are either on "Maple St." or sell "Bud" for less than $3*

✎ *Find the bars that sell the same beers with different prices.*

Relational Algebra: Complex Expr

Sells			Bars	
bar	beer	price	bar	addr
Joe's	Bud	2.50	Joe's	Maple St.
Joe's	Miller	2.75	Sue's	River Rd.
Sue's	Bud	2.50		
Sue's	Miller	3.00		
Max's	Miller	3.00		

✎ *Find the names of beers sold by "Sue's" bar*

$$\pi_{beer}(\sigma_{bar='Sue's'}(bars))$$

✎ *Find the names of all the bars that are either on "Maple St." or sell "Bud" for less than $3*

$$(\pi_{bar}((\sigma_{addr='MapleSt.'}(bars))) \cup (\pi_{bar}(\sigma_{price<3 \text{ and } beer='Bud'}(sells)))$$

✎ *Find the bars that sell the same beers with different prices.*
where sells1 = sells2 =rename(sells).

$$(\pi_{sells1.bar, sells2.bar}(\sigma_{sells1.price \neq sells2.price \text{ and } sells1.beer=sells2.beer}(sells1 \times sells2))$$

Functional Dependency

- Functional Dependency (FD):
  Let $R(\mathcal{A})$, s. t. $X \subset \mathcal{A}, Y \subset \mathcal{A}, X \cap Y = \phi$, X determine Y: $X \rightarrow Y \in \mathcal{F}$ if

  $$\forall (a^X, a^Y) \text{ and } (a\prime^X, a\prime^Y),\ a^X = a\prime^X \rightarrow a^Y = a\prime^Y$$

  where a^X is a sequence of X values of a tuple $t \in \mathcal{T}$

  ✎ $movies(title, director, date, nationality, budget)$ $\underline{title, director} \rightarrow date, nationality, budget$

- Elementary Functional Dependency (EFD) $X \rightarrow Y$ if :

  $$\nexists X_i \subset X,\ X_i \rightarrow Y,$$

  ✎ $employees(first\_name, last\_name, category, grade, salary)$ $category, grade \rightarrow salary$
  $grade \rightarrow salary$

- Relation Key:

  If $X \cup Y = \mathcal{A}$ and $X \rightarrow Y$ is a EFD, then X is a key of R

  ✎ $(title, director)$ is a key of $movies$ relation

CentraleSupélec

Functional Dependency

- ✔ A relation could have more than one key
  - ✎ $schedule(slot, room, professor, module, group)$;
    $room, slot \rightarrow professor, module, group$

    $professor, slot \rightarrow room, module, group$
- ✔ Reasoning based on FDs: Armstrong rules (transitivity, augmentation, reflexivity, ...), transitive closure & minimal coverage

Normalization Theory

- ✔ Normalization is the process of efficiently organizing data in a database
- ✔ Eliminating redundant data; ensuring data dependencies and consistencies (after updates)
  - ➤ if *grade* $\rightarrow$ *salary*, the salary should not be duplicated for each employee with the same grade.
  - ➤ the salary should not be updated for each employee with the same grade.
- ✔ Three main normal forms are defined.

Normalization Theory

Let $R(\mathcal{A}), \mathcal{T}, X \to Y \in \mathcal{F}$ and X is a key, R is of the:

- 1st Normal Form (1NF): a value of an attribute is atomic (one indivisible value).
  - ✎ phone attribute only one value, address attribute couldn't be divided into sub-attributes (street name, city, ...)
- 2nd Normal Form (2NF): non key attributes depend fully on the key

$$\nexists X' \subset X \text{ and } \nexists Y' \subset Y \text{ , } X' \to Y' \in \mathcal{F}$$

  - ✎ $schedule(slot, room, professor, group, module)$
    $slot, room \to professor, group, module$
    $professor \to module$

    $module$ depends partially on the key (slot, room)
- 3rd Normal Form (3NF): non-key attributes depend directly on the key

$$R \text{ is 2NF and } \nexists Y' \subset Y, Y'' \subseteq Y, \; Y' \to Y'' \in \mathcal{F}$$

  - ✎ $employees(identifier, first\_name, last\_name, category, grade, salary)$
    $identifier \to first\_name, last\_name, category, grade, salary$
    $grade \to salary$

    salary depends in a transitive way on the key $identifier$.

Normalization Theory

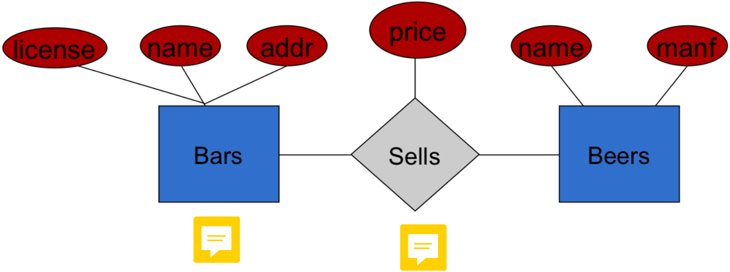
- ✔ Solution is to define a relational schema of different relations
  - ✎ Schema 1: $(slot, room), (professor, slot)$ are the relation keys
    $schedule(\underline{slot, room}, \#professor, group)$
    $professor\_module(\underline{professor}, module)$
    Or
    $schedule(\underline{slot, \#professor}, room, group)$
    $professor\_module(\underline{professor}, module)$
  - ✎ Schema 2
    $employees(\underline{identifier}, first\_name, last\_name, category, \#grade)$
    $grade\_salary(\underline{grade}, salary)$
- ✔ Underlined attributes denotes a key and # denotes a foreign key which references a key of a relation
- ✔ A foreign key value should refer to an existing key value

Normalization Theory

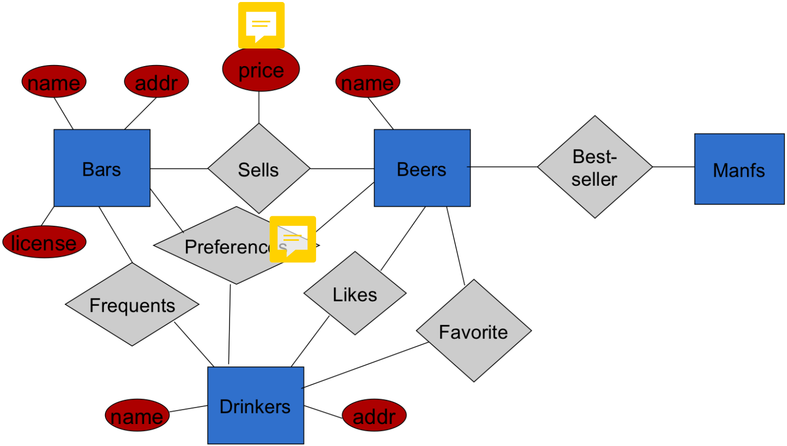
- ✔ Boyce-Codd Normal Form (BCNF): R is 3NF and $\forall X \rightarrow Y \in \mathcal{F}$, X is a Key. BCNF takes into account all candidate keys in a relation.
  - ✎ Let $R(A, B, C)$ $A, B \rightarrow C$ and $C \rightarrow B$ R is in 3NF but not in BCNF because C is not a key of R
- ✔ Decomposition algorithms to build a schema complying with 1NF, 2NF, 3NF and BCNF

Entity Association Model

- ✔ Entity defines a collection of similar entities.
- ✔ Attribute is a property of entities of an Entity.
- ✔ Association connects two (binary) or *n* (nary) kind of entities.
  - ➤ An association can also hold attributes.

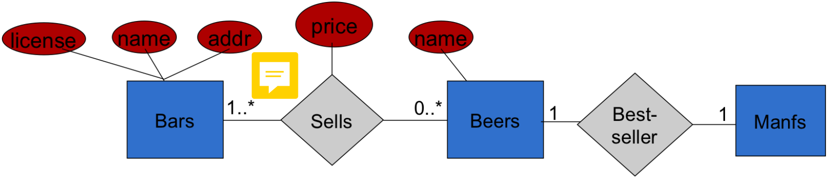


Entity Association Model



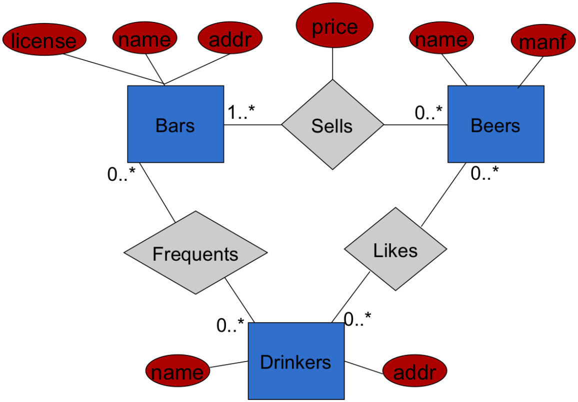
Entity Association Model

✔ Multiplicity: Associations are Many-Many or Many-One or One-One.



Mapping EA to Relational Model

- Bars(name, addr, licence); Beers (name, manf); Drinkers(name, addr); Sells((#bar, #beer, price); Likes(#drinker, #beer); Frequents(#drinker, #bar)
- The model must at least comply with the functional dependencies and be 2NF and 3NF.



Relational Algebra - SQL

- SQL is primarily a query language, for getting information from a relational database. It also includes schema data-definition.
  Relation $\Rightarrow$ table ; Tuple $\Rightarrow$ row ; Attribute $\Rightarrow$ column ; Relational model $\Rightarrow$ schema
- A bag is a collection where an element may appear more than once.
- SQL is a bag language.
  - Bag Laws $\neq$ Set Laws
    Ex. Set is idempotent for sets $S \cup S = S$, but not for bags

SQL Language

✔ Creating or deleting a table:

```
CREATE TABLE <name> (<list of declarations> );
DROP TABLE <name>;
```

✔ Declarations include:

- ✎ Column's name and its type. The most common types: `INT` or `INTEGER`; `REAL` or `FLOAT` or `NUMERIC(n, nbDecimals)`; `CHAR(n)` a fixed-length string; `VARCHAR(n)` a variable-length string of up to `n`; `DATE` or `TIME` or `TIMESTAMP` date and time.
- ✎ Constraints as Primary Key and Foreign Key

```
CREATE TABLE Beers (name CHAR(20) PRIMARY KEY,
addr VARCHAR(20), licence CHAR(10));
CREATE TABLE Bars (name CHAR(20) PRIMARY KEY, manf CHAR(20));
CREATE TABLE Sells (bar CHAR(20) REFERENCES Bars(name),
beer CHAR(20) REFERENCES Beers(name), price REAL, PRIMARY KEY (bar, beer));
```

SQL Language

- ✔ Some other constraints as: `UNIQUE`, `NOT NULL`, `CHECK`

```
CREATE TABLE Persons (
    ID int UNIQUE,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int CHECK (Age>=18)
);
```

- ✔ More complex constraints with `CHECK` and others require procedural language (conditional and repetitive instructions, variables)

SQL Language

- ✔ `SELECT-FROM-WHERE` Statements

```
SELECT name FROM Beers WHERE manf = 'Anheuser-Busch';
SELECT * FROM Beers WHERE manf = 'Anheuser-Busch';
SELECT name AS beer, manf  FROM Beers WHERE manf = 'Anheuser-Busch';
SELECT bar, beer, price*114 AS priceInYen FROM Sells;
SELECT price FROM Sells WHERE bar = 'Joe''s Bar' AND beer = 'Bud';
SELECT price FROM Sells WHERE bar like 'Joe%' AND beer = 'Bud';
```

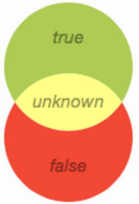
- ✔ Comparisons:

```
= , <> , != , > , < , >= , <= , IN , BETWEEN , LIKE , IS NULL , IS NOT NULL
```

SQL Language

- ✔ The logic of conditions in SQL is 3-valued logic: `TRUE`, `FALSE`, `UNKNOWN`. 2-valued laws $\neq$ 3-valued laws.

```
NULL = NULL ; NULL <> 1; 1800 + NULL > 1200 -> unknown
unknown OR false -> unknown
unknown OR true -> true
unknown AND false -> false
unknown AND true -> tunknown
SELECT col FROM t WHERE col = NULL -> unknown
SELECT col FROM t  WHERE col IS NULL -> to use
```



SQL Language

✎ Multi-table Queries

```
SELECT beer FROM Likes, Frequents
WHERE bar = 'Joe''s Bar' AND Frequents.drinker = Likes.drinker;
SELECT b1.name, b2.name FROM Beers b1, Beers b2
WHERE b1.manf = b2.manf AND b1.name < b2.name;
```

✎ Join variants

```
R CROSS JOIN S
R NATURAL JOIN S
R JOIN S ON <condition>;
R LEFT/RIGHT/FULL OUTER JOIN S ON <condition>
```

SQL Language

✎ Bars(name, addr, licence); Beers (name, manf); Drinkers(name, addr);
Sells((#bar, #beer, price); Likes(#drinker, #beer); Frequents(#drinker, #bar)

✎ A parenthesized `SELECT-FROM-WHERE` statement (sub-queries) can be used as a value in some clauses, including `FROM` and `WHERE` clauses.

```
SELECT beer FROM Likes,
(SELECT drinker FROM Frequents WHERE bar = 'Joe"s Bar')JD
WHERE Likes.drinker = JD.drinker;
```

```
SELECT bar FROM Sells WHERE beer = 'Miller'
AND price =  (SELECT price  FROM Sells WHERE bar = 'Joe"s Bar'  AND beer = 'Bud');
```

```
SELECT * FROM Beers WHERE name IN (SELECT beer
FROM Likes WHERE drinker = 'Fred');
```

```
SELECT name, man FROM Beers b WHERE NOT EXISTS (SELECT *
FROM Sells WHERE beer = b.name);
```

`EXISTS` returns true if at least one result

```
SELECT beer FROM Sells WHERE price >= ALL(SELECT price FROM Sells);
```

`ALL` returns true if the condition is true for all results

```
SELECT beer, bar  FROM Sells WHERE beer = ANY(SELECT beer FROM  Likes);
```

`ANY` returns true if the condition is true for any

SQL Language

- ✔ Intersection, Union, Difference

```
(SELECT * FROM Likes)
INTERSECT
(SELECT drinker, beer FROM Sells, Frequents WHERE Frequents.bar = Sells.bar);
```

- ✔ Selected columns must be of similar types
- ✔ Duplicates are by default eliminated
- ✔ Force the result to be a bag by `ALL` as `UNION ALL`

SQL Language - Aggregation Operators -

- ✔ Aggregation operators applied to entire column values of a table, produce a single result. Ex. `SUM, AVG, COUNT, MIN, MAX`.
  - ✎ `NULL` values are ignored in an aggregation.
  - ✎ `COUNT(col)` returns the count of non-`NULL` values of `col`. `COUNT(*)` total number of rows of a table.
  - ✎ If there are no non-`NULL` values in a `col`, the result of an aggregation is `NULL` except `COUNT(col)` returns 0.
  - ✎ Use `DISTINCT` inside an aggregation to eliminate duplicates.

```
SELECT AVG(price) FROM Sells WHERE beer = 'Bud';
SELECT COUNT(DISTINCT price) FROM Sells WHERE beer = 'Bud';
SELECT bar, MIN(price) FROM Sells WHERE beer = 'Bud' -> illegal
```

SQL Language - GROUP BY Clause -

✔ `GROUP BY <list of cols>` clause aggregates rows of list columns having the same values and produces a groups for each value.

- ✎ With `GROUP BY`, rows with `NULL` values go into one group, and the aggregates are computed for this group, as for any other.
- ✎ Projected columns in a `GROUP BY` must either appear in the `GROUP BY` clause or under an aggregate function.

```
SELECT beer, AVG(price) FROM Sells GROUP BY beer;

SELECT AVG(price) FROM Sells GROUP BY bar ;

SELECT drinker, AVG(price)
FROM Frequents, Sells
WHERE beer = 'Bud' AND Frequents.bar = Sells.bar
GROUP BY drinker;
```

✎ Bars(name, addr, licence); Beers (name, manf); Drinkers(name, addr); Sells((#bar, #beer, price); Likes(#drinker, #beer); Frequents(#drinker, #bar)

SQL Language - HAVING Clause -

- ✔ `HAVING <condition>` clause may follow a `GROUP BY` and applies the `<condition>` to each group, groups not satisfying the `<condition>` are eliminated.
- ✔ `HAVING <condition>` without `GROUP BY` operates on all-at-once the table as a set

```
SELECT beer, AVG(price) FROM Sells GROUP BY beer HAVING AVG(price) > =3
-> average computed by beer
```

```
SELECT beer, AVG(price) FROM Sells
GROUP BY beer
HAVING COUNT(bar) >= 3 OR beer IN (SELECT name FROM Beers WHERE manf = 'Pete''s')
```

✎ Bars(name, addr, licence); Beers (name, manf); Drinkers(name, addr);
Sells((#bar, #beer, price); Likes(#drinker, #beer); Frequents(#drinker, #bar)

SQL Language - INSERT -

✎ Bars(<u>name</u>, addr, licence); Beers (<u>name</u>, manf); Drinkers(<u>name</u>, addr);
Sells(<u>(#bar, #beer</u>, price); Likes(<u>#drinker, #beer</u>); Frequents(<u>#drinker, #bar</u>)

✎ Insertion

```
INSERT INTO <relation> VALUES ( <list of values> );
INSERT INTO Likes VALUES('Sally', 'Bud');
INSERT INTO Likes(beer, drinker) VALUES('Bud', 'Sally');

INSERT INTO <relation> ( <subquery> );
INSERT INTO PotBuddies (SELECT d2.drinker FROM Frequents d1, Frequents d2
WHERE d1.drinker = 'Sally' AND d2.drinker <> 'Sally' AND d1.bar = d2.bar);
```

SQL Language - DELETE -

✎ Bars(name, addr, licence); Beers (name, manf); Drinkers(name, addr);
Sells((#bar, #beer, price); Likes(#drinker, #beer); Frequents(#drinker, #bar)

✎ Deletion

```
DELETE FROM <relation> WHERE <condition>;
DELETE FROM Likes WHERE drinker = 'Sally' AND beer = 'Bud';

DELETE FROM Beers b WHERE EXISTS (SELECT name FROM Beers WHERE manf = b.manf AND
name <> b.name);
```

```
SELECT b1.name, b2.name FROM Beers b1, Beers b2
WHERE b1.manf = b2.manf AND b1.name < b2.name;
```

SQL Language - UPDATE -

✎ Updates

```
UPDATE <relation> SET <list of attribute assignments> WHERE <condition on tuples>;

UPDATE Drinkers SET phone = '555-1212' WHERE name = 'Fred';

UPDATE Sells SET price = 4.00 WHERE price > 4.00;
```