These lecture notes include some material from Professors Bertossi, Kolaitis, Guagliardo and Libkin

Relational Data Model and Relational Algebra

Lecture Handout 3

Dr Evgenia Ternovska
Associate Professor

Simon Fraser University

Fall 2017

On Database Systems

Databases are sets of mutually related data items

- They represent some aspects of the real world
- Aspects that are relevant for a particular application or specific set of users
- Databases must be "Logically coherent"

They are accessed and manipulated by means of Database Management Systems (DBMSs)

Database Management Systems

A database management system (DBMS) provides support for:

- At least one data model (a mathematical abstraction for representing data);
- At least one high level data language (language for defining, updating, manipulating, and retrieving data);
- Transaction management & concurrency control mechanisms;
- Access control (limit access of certain data to certain users);
- Resiliency (ability to recover from crashes);
- Integrity and consistency of information;
- Concurrent access to information:
  Different transactions can simultaneously access and modify the database
- Possibility of building applications on top:
  Writing and running programs that interact with the database

Logical separation between data and programs:

In a file based system (or data models before the relational) the structure of data was embedded in the data manipulation programs

Any change in the structure of data implied change of the programs and vice versa

Relational Databases: How it Started

The history of relational databases is the history of a scientific and technological revolution.

The scientific revolution started in 1970 by Edgar (Ted) F. Codd at the IBM San Jose Research Laboratory (now the IBM Almaden Research Center)

Codd introduced the relational data model and two database query languages: **relational algebra** and **relational calculus**.

"A relational model for data for large shared data banks", CACM, 1970.

"Relational completeness of data base sublanguages, in: Database Systems, ed. by R. Rustin, 1972."

Brief History:

1961:	First DBMS: *Integrated Data Store* of GE
1962:	IBM and AA develop SABRE
1966-1969:	IBM develops *Information Management System* (IMS). Uses hierarchical model.
1970:	Edgar Codd (IBM) proposes the relational model of data, and specifies how a relational DBMS could be built accordingly
1975:	First international conferences ACM SIGMOD and VLDB
1976:	Peter Chen introduces the ER model
70s:	Development of first RDBMS: more on it on the next slide
80s:	DBMSs for PCs (DBASE, Paradox, etc.).

Relational Databases: the Early Years

In the 1970th, researchers at the IBM San Jose Laboratory embark on the System R project, the first implementation of a relational database management system (RDBMS)

In 1974-1975, they develop SEQUEL, a query language that eventually became the industry standard SQL.

System R evolved to DB2 released first in 1983.

M. Stonebraker and E. Wong embark on the development of the Ingres RDBMS at UC Berkeley in 1973.

Ingres is commercialized in 1983; later, it became PostgreSQL, a free software OODBMS (object-oriented DBMS).

L. Ellison founds a company in 1979 that eventually becomes Oracle Corporation; Oracle V2 is released in 1979 and Oracle V3 in 1983.

1981: Edgar F. Codd receives the ACM Turing Award

For his fundamental and continuing contributions to the theory and practice of database management systems.
He **originated the relational approach to database management** *in a series of research papers published commencing in 1970.*
His paper "A Relational Model of Data for Large Shared Data Banks" was a seminal paper, in a continuing and carefully developed series of papers.

The contribution had impact on numerous related areas, including database languages, query subsystems, database semantics, locking and recovery, and inferential subsystems.

About ACM Turing Award

A.M. Turing Award given by the Association for Computing Machinery

ACM's **most prestigious technical award** is accompanied by a prize of $250,000.

It is given to an individual selected for contributions of a technical nature made to the computing community.

The contributions should be of lasting and major technical importance to the computer field.

Financial support of the Turing Award is provided by the Intel Corporation and Google Inc.

http://awards.acm.org/homepage.cfm?srt=all&awd=140

Brief History (Cont.)

1985: Preliminary publication of standard for SQL. Object Oriented DBMSs. Client/server architectures. Distributed DBs.

90s: New functionalities:

- Spatial DBs: Store, manipulate, query spatial objects
- Temporal DBs: Manipulate time as we understand time, not as any other numerical attribute
- Active DBs: Store *active rules* (or triggers) in the DB that react executing actions (**A**) when certain events (**E**) inside the DB happen (e.g. updates) and when certain conditions (**C**) are satisfied (checked through an internal and fixed query) The action can be of several kinds, e.g. additional updates, message to the user/application, etc.
  ECA rules: $E,\ C\ \Rightarrow\ A$

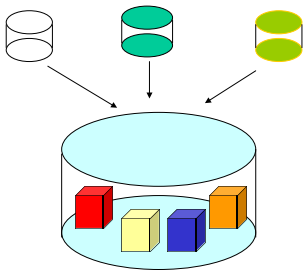
- Deductive DBs: Have deductive rules in addition to the tables, producing *implicit data* via the combination; data that can be queried and computed (by logical inference)

  Best example: recursive rules that define (and compute upon request) the transitive closure of a relation stored in the DB

- OO DBs
- Multimedia DBs
- Data Mining: Mechanisms for learning from data, detection of patterns, associations, etc.
- Workflows, etc.: Processes realizing several tasks that involve automated use of DBs

Brief History (Cont.)

- Data Warehouses: Large repositories of physically integrated data, designed for data analysis, business undertanding/learning, etc.
  Not for operational/transactional purposes; they handle complex queries (involving massive data); implemented separately from operational DB

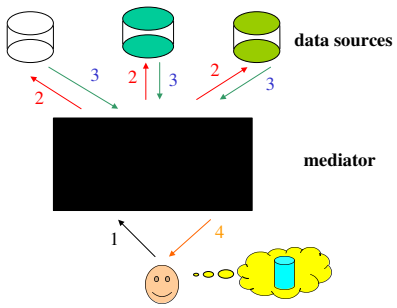


1998: Jim Gray receives the ACM Turing Award

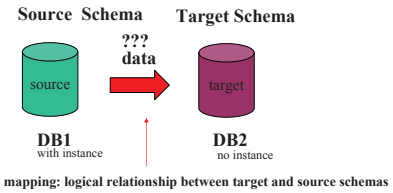
For seminal contributions to database and transaction processing research and technical leadership in system implementation.

Late 90s - approx. 2010:

- Virtual Data Integration: Mediators integrate data sources leaving data at the sources, and offering a database-like interface to users, with global schema (possible different from local schemas)
  Mediator requires and contains *mappings* between global schema and each of local schema; and a *general query planner*



- Data Exchange: Moving data from a source database to a target database that has no data and a different schema



- Interaction with the *WWW*, XML, Semantic Web, Web Services, P2P Data Management Systems, Data Streams, ...

These days:

- Big data
- Data in the cloud
- "no-SQL" DBs
- Business intelligence
- Data on the web: RDF data, linked data, graph DBs
- Data access through ontologies and conceptual models

We will focus on the Relational data model next, and we will give precise definitions

Relational Database Industry Today

According to Gartner, Inc., June 2007:

Worldwide relational database management systems (RDBMS) total software revenue totalled $15.2 billion in 2006, a 14.2 percent increase from 2005 revenue of $13.3 billion.

In 2007, the total RDBMS software revenue increased to $17.1 billion (figures released in July 2008).

In 2015, **$35.9 billion**, more than doubled compared to 2007

Data Models and Data Languages

- A data model is a mathematical formalism for describing and representing data.
- A data model is accompanied by a data language that has two parts: a data definition language and a data manipulation language.
- A **data definition language (DDL)** has a syntax for describing database templates in terms of the underlying data model.
- A **data manipulation language (DML)** supports the following operations on data:
  - Insertion
  - Deletion
  - Update
  - Retrieval and extraction of data (query the data).
- The first three operations are fairly standard. However, there is much variety on data retrieval and extraction (query languages).

The Relational Data Model (E.F. Codd 1970)

The Relational Data Model uses the mathematical concept of a relation as the formalism for describing and representing data.

Question: What is a relation?

Answer:

- Formally, a relation is a subset of a cartesian product of sets.
- Informally, a relation is a table with rows and columns.

Customer

CustID	Name	City	Address
cust1	Renton	Edinburgh	2 Wellington Pl
cust2	Watson	London	221B Baker St
cust3	Holmes	London	221B Baker St

Basic Notions from Discrete Mathematics

A k-tuple is an ordered sequence of k objects (need not be distinct)
$(2, 0, 1)$ is a 3-tuple; (a, b, a, a, c) is a 5-tuple, and so on.

If $D_1, D_2, \ldots, D_k$ are k sets, then the **cartesian product**
$D_1 \times D_2 \cdots \times D_k$ of these sets is the set of all k-tuples
$(d_1, d_2, \ldots, d_k)$ such that $d_i \in D_i$, for $1 \leq i \leq k$.

Fact: Let $|D|$ denote the cardinality ($= \#$ of elements) of a set D.
Then $|D_1 \times D_2 \times \cdots \times D_k| = |D_1| \times |D_2| \times \cdots \times |D_k|$.

Example

If $D_1 = \{0, 1\}$ and $D_2 = \{a, b, c, d\}$, then $|D_1| \times |D_2| = 8$.

Warning: Computing cartesian products is an expensive operation!

Basic Notions from Discrete Mathematics

A k-ary relation R is a subset of a cartesian product of k sets, i.e.,
$R \subseteq D_1 \times D_2 \times \cdots \times D_k$.

Example

Unary $R = \{0, 2, 4, \ldots, 100\}$ $(R \subseteq D)$
Binary $T = \{(a, b) : a$ and b have the same birthday$\}$
Ternary $S = \{(m, n, s) : s = m + n\}$
...

Relations and Attributes

Note:
$R \subseteq D_1 \times D_2 \times \ \times D_k$ can be viewed as a table with k columns

Example

Table Sales:

Customer_{25}	0	3672	45	28	3672	3	67
Customer_{2}	9	8392	88	72	7292	8	23

In the relational data model, we want to have names for the columns; these are the **attributes** of the relation.

Relation Schemas and Relational Database Schemas

A k-ary **relation schema** $R(A_1, A_2, , A_K)$ is a set $A_1, A_2, , A_k$ of k attributes.

Example

COURSE(course-no, course-name, term, instructor, room, time)
CITY-INFO(name, state, population)

Thus, a k-ary relation schema is a blueprint, a template for some k-ary relation.

An **instance** of a relation schema is a relation conforming to the schema (arities match; also, in DBMS, data types of attributes match).

A **relational database schema** is a set of relation schemas $R_i(A_1, A_2, , A_{k_i})$, for $1 \leq i \leq m$.

A **relational database instance** of a relational schema is a set of relations R_i each of which is an instance of the relation schema R_i, $1 \leq i \leq m$.

Relational Database Schemas - Examples

BANKING relational database schema with relation schemas

CHECKING-ACCOUNT(branch, acc-no, cust-id, balance)
SAVINGS-ACCOUNT(branch, acc-no, cust-id, balance)
CUSTOMER(cust-id, name, address, phone, email)
....

UNIVERSITY relational database schema with relation schemas

STUDENT(student-id, student-name, major, status)
FACULTY(faculty-id, faculty-name, dpt, title, salary)
COURSE(course-no, course-name, term, instructor)
ENROLLS(student-id, course-no, term)
....

Schemas vs. Instances

Keep in mind that there is a **clear distinction** between

- relation schemas and instances of relation schemas

and between

- relational database schemas and relational database instances.

Important Difference:

Syntactic Notion	**Semantic Notion (discrete mathematics notion)**
Relation Schema	Instance of a relation schema (i.e., a relation)
Relational Database Schema	Relational database instance (i.e., a database)

Programming Languages Paradigms

There are two main paradigms of programming languages: **imperative** (or procedural) languages and **declarative** languages.

Imperative (Procedural) Languages: programs are expressed by specifying how the task is to be accomplished (sequence of operations).

FORTRAN, C, ...

Declarative Languages: programs are expressed by specifying what has to be accomplished (as opposed to how).

LISP (functional programming), PROLOG (logic programming), ...

Query Languages for the Relational Data Model

Codd introduced two different query languages for the relational data model:

Relational Algebra, which is a procedural language.
It is an algebraic formalism in which queries are expressed by applying a sequence of operations to relations.

Relational Calculus, which is a declarative language.

It is a logical formalism in which queries are expressed as formulas of first-order logic.

Codds Theorem: Relational Algebra and Relational Calculus are essentially equivalent in terms of expressive power.

(but what does this really mean?)

Desiderata for a Database Query Language

The language should be sufficiently high-level to secure physical data independence, i.e., the separation between the physical level and the conceptual level of databases.

The language should have **high enough expressive power** to be able to pose useful and interesting queries against the database.

The language should be **efficiently implementable** to allow for the fast retrieval of information from the database.

Warning:

There is a **tension** between the last two desiderata.
Increase in expressive power comes at the expense of efficiency.

Relational algebra

Relational algebra strikes a good balance between expressive power and efficiency.

Codd's key contribution was to identify a small set of basic operations on relations and to demonstrate that useful and interesting queries can be expressed by combining these operations.

- Thus, relational algebra is a rich enough language, even though, as we will see later on, it suffers from certain limitations in terms of expressive power.

The first RDBMS prototype implementations (System R and Ingres) demonstrated that the relational algebra operations can be implemented efficiently.

The Basic Operations of Relational Algebra

Group I: Three standard set-theoretic binary operations:

- Union
- Difference
- Cartesian Product.

Group II. Two unary operations on relations:

- Projection
- Selection.

Group III. One special operation:

- Renaming

Relational Algebra consists of all expressions obtained by combining these basic operations in syntactically correct ways.

Relational algebra

Procedural query language

A relational algebra expression

- takes as input one or more relations
- applies a **sequence of operations**
- returns a relation as output

Operations:

Projection	(π)	Union	$(\cup)$
Selection	(σ)	Intersection	$(\cap)$
Product	$(\times)$	Difference	$(-)$
Renaming	(ρ)		

The application of each operation results in a new relation that can be used as input to other operations

Projection

- **Vertical operation**: choose some of the columns
- Syntax: $\pi_{\text{set of attributes}}(\text{relation})$
- $\pi_{A_1,\ldots,A_n}(R)$ takes only the values of attributes $A_1, \ldots, A_n$ for each tuple in R

Customer

CustID	Name	City	Address
cust1	Renton	Edinburgh	2 Wellington Pl
cust2	Watson	London	221B Baker St
cust3	Holmes	London	221B Baker St

$\pi_{\text{Name,City}}(\text{Customer})$

Name	City
Renton	Edinburgh
Watson	London
Holmes	London

Selection

- **Horizontal operation**: choose rows satisfying some condition
- Syntax: $\sigma_{\text{condition}}(\text{relation})$
- A family of unary operations, one for each condition Θ
- $\sigma_\theta(R)$ takes only the tuples in R for which θ is satisfied

$$\begin{aligned} \text{term} &:= \text{attribute} \mid \text{constant} \\ \theta &:= \text{term } \textbf{op } \text{term with } \textbf{op} \in \{=, \neq, >, <, \geqslant, \leqslant\} \\ &\mid \theta \wedge \theta \mid \theta \vee \theta \mid \neg\theta \end{aligned}$$

Example of selection

Customer

CustID	Name	City	Age
cust1	Renton	Edinburgh	24
cust2	Watson	London	32
cust3	Holmes	London	35

$\sigma_{\text{City}\neq\text{'Edinburgh'} \wedge \text{Age}<33}(\text{Customer})$

CustID	Name	City	Age
cust2	Watson	London	32

More on the Selection Operator

Note: The use of the comparison operators $<$, $>$, $\leq$, $\geq$ assumes that the underlying domain of values is **totally ordered**.

If the domain is not totally ordered, then only $=$ and $\neq$ are allowed.

If we do not have attribute names (hence, we can only reference columns via their component number), then we need to have a special symbol, say \$, in front of a component number.

Thus, $\$4 > 100$ is a meaningful basic clause $\$1 =$"Apto" is a meaningful basic clause, and so on.

Efficiency (1)

Consecutive selections can be combined into a single one:

$$\sigma_{\theta_1}\big(\sigma_{\theta_2}(R)\big) = \sigma_{\theta_1 \wedge \theta_2}(R)$$

Example

$Q_1 = \sigma_{\text{City}\neq\text{'Edinburgh'}}\big(\sigma_{\text{Age}<33}(\text{Customer})\big)$

$Q_2 = \sigma_{\text{City}\neq\text{'Edinburgh'} \wedge \text{Age}<33}(\text{Customer})$

$Q_1 = Q_2$ but Q_2 **faster** than Q_1 in general

Efficiency (2)

Projection can be pushed inside selection

$$\pi_\alpha\big(\sigma_\theta(R)\big) = \sigma_\theta\big(\pi_\alpha(R)\big)$$

only if all attributes mentioned in θ appear in α

Example

$Q_1 = \pi_{\text{Name,City,Age}}\big(\sigma_{\text{City}\neq\text{'Edinburgh'} \wedge \text{Age}<33}(\text{Customer})\big)$

$Q_2 = \sigma_{\text{City}\neq\text{'Edinburgh'} \wedge \text{Age}<33}\big(\pi_{\text{Name,City,Age}}(\text{Customer})\big)$

Question: Which one is more efficient?

Cartesian product

$R \times S$ **concatenates** each tuple of R with all the tuples of S

Example

R	**A**	**B**
	1	2
	3	4

$\times$

S	**C**	**D**
	1	a
	2	b
	3	c

$=$

$R \times S$	**A**	**B**	**C**	**D**
	1	2	1	a
	1	2	2	b
	1	2	3	c
	3	4	1	a
	3	4	2	b
	3	4	3	c

Expensive operation:

- $\text{card}(R \times S) = \text{card}(R) \times \text{card}(S)$
- $\text{arity}(R \times S) = \text{arity}(R) + \text{arity}(S)$

Joining relations

Combining Cartesian product and selection

Customer: ID, Name, City, Address
Account: Number, Branch, CustID, Balance

We can join customers with the accounts they own as follows

$$\sigma_{\text{ID}=\text{CustID}}(\text{Customer} \times \text{Account})$$

Renaming

Gives a new name to some of the attributes of a relation

Syntax: $\rho_{\text{replacements}}(\text{relation})$,
where a replacement has the form $A \to B$

$$\rho_{A \to A', C \to D} \left(\begin{array}{|c|c|c|} \hline \textbf{A} & \textbf{B} & \textbf{C} \\ \hline \text{a} & \text{b} & \text{c} \\ 1 & 2 & 3 \\ \hline \end{array} \right) = \begin{array}{|c|c|c|} \hline \textbf{A}' & \textbf{B} & \textbf{D} \\ \hline \text{a} & \text{b} & \text{c} \\ 1 & 2 & 3 \\ \hline \end{array}$$

Example

Customer: **CustID**, Name, City, Address
Account: Number, Branch, **CustID**, Balance

$$\sigma_{\text{CustID}=\text{CustID}'}\left(\text{Customer} \times \rho_{\text{CustID} \to \text{CustID}'}(\text{Account})\right)$$

Natural join

Joins two tables on their **common attributes**

Example

Customer: **CustID**, Name, City, Address
Account: Number, Branch, **CustID**, Balance

Customer $\bowtie$ Account $=$

$$\pi_{X \cup Y}\left(\sigma_{\text{CustID}=\text{CustID}'}\left(\text{Customer} \times \rho_{\text{CustID} \to \text{CustID}'}(\text{Account})\right)\right)$$

where $X = \{$ all attributes of Customer $\}$
$Y = \{$ all attributes of Account $\}$

Set operations

Union

R	**A**	**B**
	a1	b1
	a2	b2

$\cup$

S	**A**	**B**
	a1	b1
	a3	a3

$=$

R ∪ S	**A**	**B**
	a1	b1
	a2	b2
	a3	b3

Intersection

R	**A**	**B**
	a1	b1
	a2	b2

$\cap$

S	**A**	**B**
	a1	b1
	a3	a3

$=$

R ∩ S	**A**	**B**
	a1	b1

Difference

R	**A**	**B**
	a1	b1
	a2	b2

$-$

S	**A**	**B**
	a1	b1
	a3	a3

$=$

R − S	**A**	**B**
	a2	b2

The relations must have the same set of attributes

Union and renaming

R	**Father**	**Child**
	George	Elizabeth
	Philip	Charles
	Charles	William

S	**Mother**	**Child**
	Elizabeth	Charles
	Elizabeth	Andrew

We want to find the relation **parent-child**

$\rho_{\text{Father}\to\text{Parent}}(\text{R}) \cup \rho_{\text{Mother}\to\text{Parent}}(\text{S})$ $=$

Parent	**Child**
George	Elizabeth
Philip	Charles
Charles	William
Elizabeth	Charles
Elizabeth	Andrew

Full relational algebra

Primitive operations: π , σ , $\times$, ρ , $\cup$, $-$

Removing any of these results in a **loss of expressive power**

Derived operations

- $\bowtie$ can be expressed in terms of π , σ , $\times$, ρ
- $\cap$ can be expressed in terms difference:

$$R \cap S = R - (R - S)$$

Other derived operations

Theta-join	$R \bowtie_\theta S = \sigma_\theta(R \times S)$
Equijoin	$\bowtie_\theta$ where θ is a **conjunction of equalities**
Semijoin	$R \ltimes_\theta S = \pi_X\big(R \bowtie_\theta S\big)$ where X is the set of attributes of R
Antijoin	$R \,\overline{\ltimes}_\theta\, S = R - (R \ltimes_\theta S)$

Why use these operations?

- to write things more succintly
- they can be optimized independently

Division

R over set of attributes X

S over set of attributes $Y \subset X$

Let $Z = X - Y$

$$R \div S = \{\, \bar{r} \in \pi_Z(R) \mid \forall \bar{s} \in S \; (\bar{r}\bar{s} \in R \,)\}$$

$$= \{\, \bar{r} \in \pi_Z(R) \mid \{\bar{r}\} \times S \subseteq R \,\}$$

$$= \pi_Z(R) - \pi_Z\big(\pi_Z(R) \times S - R\big)$$

Division: Example

Exams

Student	**Course**
John	Databases
John	Chemistry
Mary	Programming
Mary	Math
Mary	Databases

CS

Course
Databases
Programming

Exams $\div$ CS $=$

Student
Mary

$$= \pi_{\textbf{Student}}(\text{Exams}) - \pi_{\textbf{Student}}\big(\pi_{\textbf{Student}}(\text{Exams}) \times \text{CS} - \text{Exams}\big)$$

Find the names of students who has taken exams in all CS courses

Relational Algebra Expression

Definition: A relational algebra **expression** is a string obtained from relation schemas using union, difference, cartesian product, projection, selection and renaming.

Context-free grammar for relational algebra expressions:

$$E := R, S, \ldots \mid (E_1 \cup E_2) \mid (E_1 - E_2) \mid (E_1 \times E_2) \mid \pi L(E) \mid \sigma_\Theta(E) \mid \rho\, E,$$

where
$R, S, \ldots$ are relation schemas
L is a list of attributes
Θ is a condition.

Strength from Unity and Combination

By itself, each basic relational algebra operation has limited expressive power, as it carries out a specific and rather simple task.

When used in combination, however, the basic relational algebra operations can express interesting and, quite often, rather complex queries.

Derived relational algebra operations are operations on relations that are expressible via a relational algebra expression (built from the basic operators).

Independence of the Basic Relational Algebra Operations

Question: Are all the basic relational algebra operations really needed? Can one of them be expressed in terms of the other four?

Theorem: Each of the basic relational algebra operations is independent of the other four, that is, it **cannot** be expressed by a relational algebra expression that involves only the other four.

Proof Idea: For each relational algebra operation, we need to discover a property that is possessed by that operation, but is not possessed by any relational algebra expression that involves only the other four operations.

Independence of the Basic Relational Algebra Operations

Proof Sketch: (projection and cartesian product only)

Property of projection: It is the only operation whose output may have arity smaller than its input.

Show, by induction, that the output of every relational algebra expression in the other four basic relational algebra is of arity at least as big as the maximum arity of its arguments.

Property of cartesian product: It is the only operation whose output has arity bigger than its input.

Show, by induction, that the output of every relational algebra expression in the other four basic relational algebra is of arity at most as big as the maximum arity of its arguments.

Exercise: Complete this proof.

Relational Algebra: Summary

When combined with each other, the basic relational algebra operations can express interesting and complex queries (natural join, quotient (division), . . .)

The basic relational algebra operations are independent of each other: none can be expressed in terms of the other.

So, in conclusion, Codds choice of the basic relational algebra operations has been very judicious.

Relational Completeness

Definition (Codd – 1972): A database query language L is **relationally complete** if it is at least as expressive as relational algebra, i.e., every relational algebra expression E has an equivalent expression F in L.

Relational completeness provides a benchmark for the expressive power of a database query language.

Every commercial database query language should be at least as expressive as relational algebra.

Exercise: Explain why SQL is relationally complete (after SQL is studied).

SQL vs. Relational Algebra

SQL	**Relational Algebra**
SELECT	Projection π
FROM	Cartesian Product $\times$
WHERE	Selection σ

Semantics of SQL via interpretation to Relational Algebra

SELECT $R_{i_1}.A_1, \ldots, R_{i_m}.A_m$
FROM $R_1, \ldots, R_k$
WHERE Ψ

$$= \pi R_{i_1}.A_1, \ldots, R_{i_m}.A_m(\sigma\Psi(R_1 \times \cdots \times R_k))$$